ADAM, Grégory
Promotion 2023
2020 – 2021

Diplôme d'ingénieur Télécom Physique Strasbourg
Spécialité Informatique et Réseaux

Mémoire de stage de 1re année

Test de performance d'un VOSpace

MANTELET
Grégory
gregory.mantelet@astro.unistra.fr

Du 14/06/2021 au 09/07/2021

(La première itération d'un mot défini dans le glossaire est suivie par un astérisque.)

Test de performance* d'un VOSpace*

Durant un stage précédent, j'ai implémenté un prototype du standard VOSpace. Ce standard précise comment un utilisateur peut accéder à des fichiers via différentes requêtes* vers le service. VOSpace est semblable à un service tel que Google drive mais spécifique à l'astronomie. Le but de ce stage était de mettre en place des tests afin de vérifier les performances de ce service. Pour mettre en place les tests de VOSpace, j'ai utilisé JMeter*, un logiciel libre permettant de simuler plusieurs utilisateurs qui accèdent au serveur en continu à une fréquence donnée et permet également de gérer des accès aléatoires. Différents scénarios* ont été mis en place, un script* a été réalisé afin d'automatiser le lancement des scénarios et la génération de rapport. Afin d'assurer une bonne reproductibilité des tests, j'ai écrit des scripts qui restaurent le contenu du VOSpace à un état initial connu et constant. Ces scripts sont lancés avant l'exécution de chaque scénario. Les rapports contiennent des graphiques et informations sur le temps de réponse ou la quantité de données échangées. Ceux-ci pourront être comparés d'un scénario à l'autre et il sera possible d'analyser comment le service réagit en fonction des différents scénarios. Tous les tests, réalisés pendant ce stage, ont permis d'analyser les performances lors d'accès très nombreux par beaucoup d'utilisateurs, par exemple 200 utilisateurs avec des accès toutes les 10ms. Cela nous a permis de détecter des problèmes de concurrence* lors du téléchargement d'un fichier par plusieurs utilisateurs, et ainsi montrer que le prototype n'était pas fiable dans ce cas. Les limites du serveur ont également pu être testées en lançant des tests avec un nombre d'utilisateurs qui augmente progressivement. Ces scénarios et scripts ont été réalisés de manière à pouvoir être utilisés pour tester d'autres VOSpaces et ainsi les comparer entre eux.

VOSpace performance testing

During my previous internship, I implemented a prototype of the VOSpace standard. This standard specifies how users can access to files using different requests to the service. VOSpace is similar to a service like Google drive but specific to astronomy. The goal of this internship was to setup tests to verify the performances of this service. To setup the tests of VOSpace, I used JMeter, a free software allowing you to simulate several users which access a same server at the same time continuously at a given frequency. This can also manage random access. Different scenarii have been setup and a script has been made to automate the execution of scenarii and generating reports. In order to guarantee a good test reproductivity, I wrote scripts restoring the content of the VOSpace to a known and constant initial state. These scripts are run before the execution of each scenario. The reports contain graphs and data about the response time and the amount of exchanged data. Comparing the report of each scenario let us analyze how the service behaves in normal but also in some extreme situations. All tests performed during this internship allowed us to analyse performances of multiple accesses with many users, for instance 200 users with an access every 10ms. This gave us the information that there was a concurrency problem when several users download the same file at the same time, and thus, it demonstrated that the prototype is not viable in this case. The limits of the server were also tested by starting tests with a few users that grows gradually. These scenarios and scripts have been made so we can use them to test other VOSpace services and so to compare them.

Table des matières

Introduction

Dans l'astronomie le partage et l'échange de données est une des démarches importantes. Un des secteurs importants pour l'Observatoire de Strasbourg est de partager des données. Il existe un standard qui permet de préciser comment échanger des données liées à l'astronomie de manière universelle. Durant un stage précédent un prototype de ce standard d'échange de données a été implémenté. Ce standard précise comment des données astronomiques peuvent être échangées sans tenir compte de la manière dont elles sont stockées.

Un aspect important d'un service comme celui-ci est son comportement lors d'un grand nombre de connexions simultanées. Il faut qu'il puisse répondre rapidement, qu'il soit résilient et reste fiable. Il est parfois plus intéressant d'avoir une réponse rapide donnant une erreur qu'attendre longtemps pour avoir une réponse qui n'est pas forcément complète ou une erreur.

Le but de mon stage a été de tester les performances de ce prototype. Les tests de performance permettent de vérifier comment réagit un service lors d'un grand nombre de connexions simultanées. Ces tests pourront montrer les faiblesses et les forces du prototype.

Scripter ces tests est un autre aspect du stage qui permet de simplifier et rendre plus rapide leur lancement. Ces tests permettront d'avoir une idée sur comment il réagit et s'il est fiable dans un environnement proche de la réalité ou extrême.

Un autre prototype est en cours de conception, il sera alors possible ultérieurement de comparer les performances des deux prototypes ainsi que leur fiabilité.

1 Présentation de l'entreprise

1.1 Historique

L'Observatoire astronomique de Strasbourg est un établissement de recherche et d'enseignement fondé en 1881. Situé sur le campus de l'Université de Strasbourg, il est une unité mixte de recherche du CNRS (Centre National de la Recherche Scientifique) et de l'Université de Strasbourg, il dépend donc de ces deux organismes. Le CDS, service dans lequel j'ai effectué mon stage porte ce nom en raison de sa première fonction, le stockage, la distribution et le traitement des données liées aux étoiles (Centre de Données Stellaire) puis devient Centre de Données astronomiques de Strasbourg lorsqu'il commence à élargir son domaine à d'autres types d'objets astronomiques.

1.2 Structure et domaines d'activités

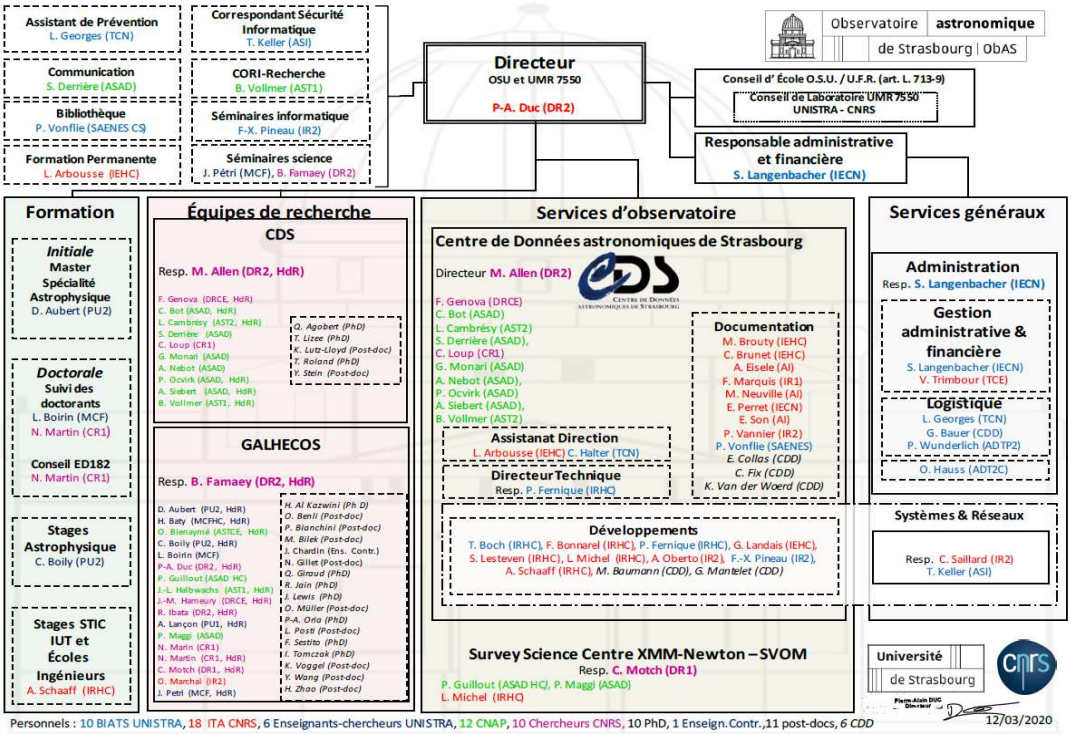
La principale fonction l'Observatoire astronomique de Strasbourg n'est pas l'observation du ciel mais la distribution, l'analyse et le traitement de données astronomiques. Il est composé de deux équipes scientifiques :

GALHECOS (*Galaxies, High Energy, Cosmology, Compact Objects & Stars*) :

Cette équipe étudie les galaxies, notamment leur formation et évolution.

Le CDS (Centre de Données astronomiques de Strasbourg) :

Le principal objectif du CDS est de répertorier, stocker et distribuer des données astronomiques extrasolaires en proposant des services permettant l'accès et l'utilisation de ces données, celles-ci pouvant être des images, des liens vers des articles ou encore des catalogues d'objets astronomiques. Il s'occupe également d'effectuer des recherches à l'aide de ces données. Ainsi cette équipe doit pouvoir gérer un grand nombre de données que cela soit en termes de stockage, traitement ou encore pouvoir répondre aux requêtes faites par les utilisateurs.
L'Observatoire compte 87 personnes dont 38 au sein du CDS. Le directeur de l'Observatoire est Pierre-Alain DUC.



Cet établissement de recherche est aussi membre de l'IVOA (*International Virtual Observatory Alliance*). Il s'agit d'une organisation scientifique composée de plusieurs organismes du monde entier. Sa principale mission est de définir des standards afin d'assurer une interopérabilité entre les services de ces organismes. L'ensemble des services issus de cette collaboration constitue ce que l'on appelle l'Observatoire Virtuel.

Figure 2 : Logo de l'IVOA

Les services

Le CDS propose plusieurs services à la communauté astronomique et au grand public :

SIMBAD :

Figure 3 : Logo SIMBAD

Une base de données d'objets astronomiques contenant des informations sur des objets hors du système solaire. Elle est mise à jour quotidiennement par identification croisée. Cela permet de pouvoir retrouver un objet en utilisant n'importe quel identifiant le représentant. Il est également possible de retrouver un objet en utilisant sa position dans le ciel. Les documentalistes doivent alors regrouper toutes les informations concernant le même objet en prêtant attention aux identifiants qui doivent correspondre au bon objet. Cette base compte plus de 10 millions d'objets astronomiques et plus de 35 millions d'identifiants.

VizieR :

Figure 4 : Logo VizieR

Une base de données de catalogues astronomiques. Un catalogue est un ensemble de données provenant d'observations effectuées au sol ou depuis l'espace, et stocké sous la forme d'une ou plusieurs tables. Il permet aux utilisateurs d'extraire des informations en utilisant certains critères de sélection. VizieR répertorie actuellement 19765 catalogues.

Aladin :

Figure 5 : Logo Aladin

C'est un atlas interactif du ciel réalisé en Java. Il permet de visualiser et localiser des objets du ciel en précisant un nom d'objet. Il est possible de zoomer sur ces objets et d'appliquer différents filtres. SIMBAD et VIZIER peuvent être couplés avec Aladin afin d'afficher des informations de ces bases en plus des images d'Aladin. Il existe aussi une version web codée en Javascript appelée Aladin lite mais avec un nombre réduit de fonctionnalités.

1.3 Travailler à l'Observatoire

Mon poste au sein de l'Observatoire été d'assister les ingénieurs dans leurs travaux et plus spécifiquement mon tuteur Grégory MANTELET. Ici mettre en place des tests sur un prototype spécifique de service. Les échanges se déroulaient en visio-conférence environ une fois par semaine au début du stage, car étant entièrement en distanciel à ce moment. Puis en présentiel lorsqu'il est devenu possible de l'être deux fois par semaine. Ces réunions permettaient de discuter des avancés ou découvertes, ou encore les problèmes rencontrés et de pouvoir prévoir ce qu'il faudra faire pour la suite et les modifications à apporter.

2 Mission du stage

2.1 Le standard VOSpace

Le standard VOSpace est une interface* mise en place par l'IVOA* qui spécifie comment accéder à des stockages de données afin de conserver, modifier et récupérer des informations. Cela permet d'avoir une manière unique et universelle pour mettre en place un système d'échange de données : le client n'a pas à faire attention à la manière dont une donnée est stockée ; la réponse à la requête d'un client sera la même quel que soit le système de stockage derrière l'interface. Grâce à ce service un client peut :

- Ajouter ou supprimer des éléments.
- Manipuler et accéder à des métadonnées* telles que : une description, un sujet.
- Accéder à des données via un URI*

Chaque donnée est représentée sous la forme d'un nœud*. À un nœud sont associées des métadonnées descriptives.

Le modèle défini par IVOA pourrait être comparé à un système de fichier comme dans n'importe quel système d'exploitation comme par exemple : Windows, MacOS ou encore Linux, un nœud dans VOSpace représentant un fichier, un répertoire ou un lien vers un autre nœud.

Ce standard recommande aussi une manière asynchrone* d'effectuer les transferts et certaines manipulations de données. Cela signifie, par exemple, que lors d'une copie ou d'un déplacement d'un nœud, le client n'a pas besoin de rester connecté au service jusqu'à ce que l'action soit terminée. De même lors d'un transfert (téléchargement), la génération de l'URL vers laquelle le client pourra transférer des données pourra se faire de manière asynchrone mais

le transfert en lui-même vers l'URL générée sera fait en synchrone*. Pour ce faire, VOSpace s'appuie sur UWS* (*Universal Worker Service*), un autre standard de l'IVOA qui définit comment gérer l'exécution d'actions de manière asynchrone. Ces types de transferts sont donc effectués en plusieurs étapes.

2.2 Les tests de performance

Les tests de performance ont pour but de vérifier les performances d'un service. Ces tests peuvent être distingués en plusieurs groupes notamment :

- Les tests de performance : ces tests vérifient les performances d'un service avec une charge donnée c'est-à-dire avec un nombre d'utilisateurs fixe qui effectuent des requêtes à une fréquence fixe. Une fois les tests réalisés, on vérifie si le temps de réponse, ou éventuellement d'autres paramètres, sont satisfaisants et que le serveur fonctionne correctement.
- Les tests aux limites : ces tests consistent à avoir un nombre très élevé d'utilisateurs virtuels et des requêtes à une fréquence elle aussi élevée pour chaque utilisateur. Ces tests permettent de déterminer comment le système réagit dans des cas plus extrêmes sur une durée donnée. Ils ne cherchent pas vraiment une limite mais teste si le service tient la charge sur des cadences très hautes, qui peuvent être au-dessus de la charge maximum réelle qu'un service peut attendre à avoir.
- Les tests de montée en charge : ces tests permettent d'avoir une idée du maximum de requêtes et/ou utilisateurs qu'un service peut supporter. L'idée est de commencer avec un nombre assez faible d'utilisateurs simulés puis d'augmenter progressivement ce nombre afin de trouver le seuil à partir duquel le service ne peut plus suivre ou que le temps de réponse n'est plus acceptable par rapport aux demandes.

Pour le prototype du VOSpace, une combinaison de ces groupes de tests a été mise en place. Les tests de performance et ceux aux limites sont confondus en faisant plusieurs tests avec un nombre fixe d'utilisateurs, dans notre cas 200. Différents tests, avec des requêtes à des fréquences différentes, sont réalisés, par exemple 10ms, 100ms et 1s. Ainsi il est possible d'observer comment fonctionne le service en cas de grosse charge ainsi que comparer comment évoluent les performances en augmentant la fréquence des requêtes. Les tests de montée en charge eux sont mis en place en augmentant de manière linéaire sur l'intervalle de temps le nombre d'utilisateurs (jusqu'à 5000 utilisateurs simultanés). Ces utilisateurs font des requêtes toutes les millisecondes pour stresser autant que possible le système. L'image ci-dessous montre un exemple de l'augmentation des 5000 utilisateurs sur une minute puis reste à 5000 jusqu'à la fin du test.

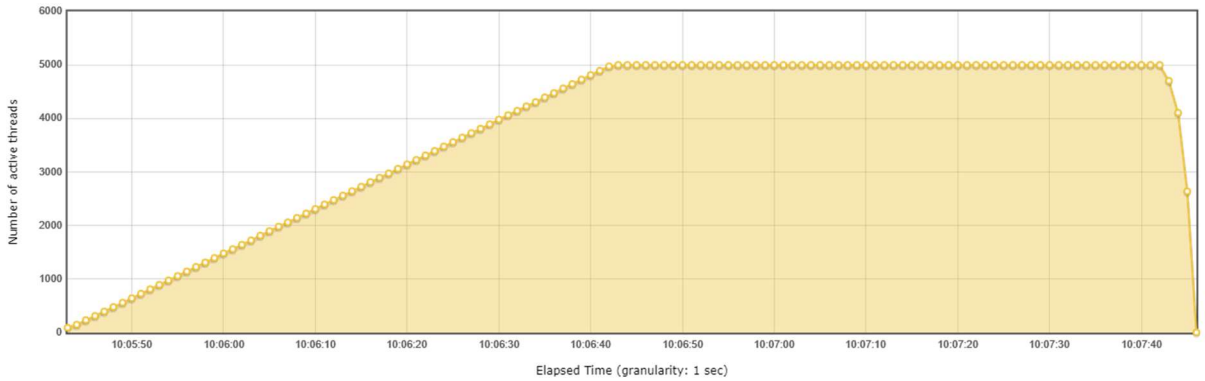


Figure 6 : Augmentation du nombre de d'utilisateur en fonction du temps

Enfin il est également possible de comparer es réactions du service en fonction du type de requête envoyé. Cela peut par exemple se faire en comparant les temps de réponse avec des paramètres de test fixes (ex: fréquence d'envoi de requête, nombre d'utilisateurs).. Les différents types de requête peuvent être : un accès, une modification de métadonnée ou encore le téléchargement d'un fichier. Ainsi, il est possible de savoir quelles sont les requêtes les plus lourdes en temps de réponse ou d'autres critères.

2.3 Qu'est-ce que JMeter ?

Apache JMeter est un logiciel Open Source permettant d'effectuer des tests sur un service, notamment des tests de performance c'est-à-dire de tester le temps de réponse ou l'utilisation de la bande passante d'un service sous certaines conditions. Par exemple simuler un grand nombre d'utilisateurs simultanés qui utilisent un service. Cela peut être accéder à des données identiques ou à de multiples données différentes. Les données peuvent être une image ou un fichier formaté (XML* par exemple).

Site de JMeter : https://jmeter.apache.org/

Figure 7: Logo de JMeter

2.3.1 Pourquoi JMeter ?

Lors des recherches pour trouver une manière d'effectuer des tests de performance sur le VOSpace, j'ai trouvé plusieurs logiciels différents pouvant réaliser des tests de performance. Notamment : SoapUI, Katalon studio et JMeter. Les deux premiers proposent une version gratuite et une autre sous licence alors que JMeter est totalement libre. De plus, les versions libres de SoapUI et Katalon studio ne donne pas la possibilité d'effectuer les tests en ligne de commande et ainsi de pouvoir scripter l'exécution des tests facilement. Ils contiennent également d'autres limitations bien que ceux-ci offrent d'autres avantages comme la visualisation en temps réel mais ces avantages restent négligeables car pouvoir effectuer des tests en ligne de commande permet notamment de pouvoir mettre en place un système de CI/CD*, c'est-à-dire d'intégration continue, donc de tester le service à chaque version automatiquement lors de sa mise en production. De plus JMeter permet également de lancer les tests en mode distribué c'est-à-dire d'utiliser la puissance de calcul de plusieurs ordinateurs coordonnés pour lancer les requêtes afin du pouvoir simuler une charge plus conséquente. De plus, pour JMeter, beaucoup de documentations et aides sont disponibles. Il permet aussi d'ajouter des *plugins** afin d'étendre les capacités du logiciel de base pour entre autres examiner l'état d'utilisation du processeur ou la mémoire du serveur.

2.3.2 Fonctionnement de JMeter

L'interface de JMeter se présente comme ci-dessous

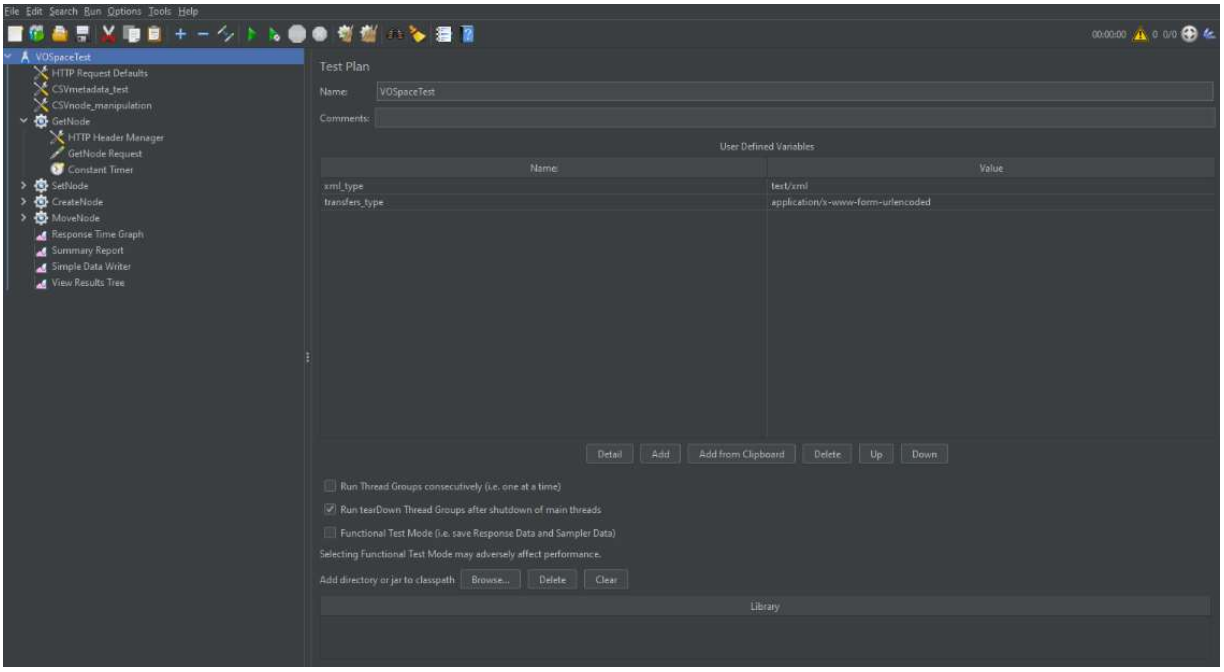


Figure 8: Interface de JMeter

Le panneau à gauche permet d'ajouter des composants qui permettront notamment de configurer l'accès au serveur : nom de domaine*, port*… Mais aussi d'ajouter des récepteurs qui récupèrent des informations sur la requête tel que le temps de réponse. Les principaux types de composant que j'ai utilisés sont :

- Les configurations : Ce sont des éléments qui préparent les requêtes et mettent en place le contexte. Il peut s'agir de spécifier un nom de domaine, un port, un protocole* commun à toutes les requêtes. Les configurations permettent aussi de charger des fichiers qui pourront être lus et utilisés lors des requêtes vers le service ; par exemple un fichier csv*. Il est possible d'initialiser des variables qui elles aussi pourront être utilisées dans tout le scénario.
- Les récepteurs : Ils permettent d'écouter les requêtes vers le serveur et ses réponses. Ainsi il est possible de récupérer des informations sur le déroulement de celle-ci (par exemple : l'utilisation du réseau, temps de latence). C'est grâce à ces composants qu'il est possible d'analyser les résultats et d'avoir des renseignements sur l'état du service lors du scenario.
- Les moteurs d'utilisateur : C'est le cœur de JMeter, ils servent à simuler un groupe d'utilisateurs qui exécuteront la suite d'instructions qui leur est donnée. C'est ici que l'on peut préciser le nombre d'utilisateurs du groupe, le temps durant lequel ils lancent des requêtes ou le temps pris pour que tous les utilisateurs soient actifs.
- Les échantillons : Ces éléments précisent la requête que les utilisateurs du moteur utiliseront. Il peut y en avoir plusieurs par moteur. Sans ces éléments, il n'y a aucune action vers un service qui peut être réalisée. Il est possible d'ajouter d'autres informations de configuration dans cet élément.
- Les compteurs de temps : Ils gèrent le temps, particulièrement le temps entre chaque requête. Il est possible d'avoir des compteurs qui gèrent le temps de manière aléatoire avec différentes répartitions (uniforme, gaussien) afin de pouvoir ajouter de l'aléatoire dans les tests pour s'approcher d'un cas réel d'utilisation.

- Les postprocesseurs : Ils sont employés afin d'analyser et extraire des informations dans la réponse. Cela peut être pour vérifier la présence de certains motifs ou des informations spécifiques et de poursuivre avec les autres instructions en fonction de ce résultat.
- Les contrôleurs logiques : Tout comme dans un langage de programmation, ils ajoutent des conditions, des boucles ainsi que d'autres types de logique afin de gérer les requêtes et d'exécuter certaines requêtes seulement dans des cas spécifiques ou d'attendre une réponse particulière du serveur et de réagir en fonction de celle-ci.

JMeter donne des recommandations pour le bon déroulement des tests. Utiliser le moins de récepteurs possible afin de ne pas dégrader les performances de la machine qui lance les requêtes au serveur. Ils déconseillent également d'utiliser certains récepteurs qui sont trop gourmands en consommation processeur. Jmeter recommande aussi de lancer les tests en ligne de commande, hormis pour le débogage*, toujours pour des questions de performance. Pour les mêmes raisons, il ne faut sauvegarder que les données nécessaires à l'analyse et que cela soit le moins d'information possible. Un autre aspect qui peut limiter l'efficacité de la machine faisant les requêtes au serveur est l'aléatoire, il est aisni plus performant d'utiliser un fichier pseudo-aléatoire pré-généré comme un fichier csv que d'utiliser les fonctions aléatoires de JMeter.

Ci-dessous un exemple de scenario simple. Le premier élément donne les informations en commun pour toutes les requêtes (nom de domaine, port). Le deuxième sert de variable pseudo-aléatoire en lisant un fichier csv, cette variable passe à la valeur suivante du csv à chaque lecture. *GetNode* est le groupe d'utilisateur. Le premier élément du groupe permet de donner des entêtes communs qui seront lus par le serveur pour vérifier quel type de donnée il reçoit et quel type il doit envoyer. La deuxième ligne du groupe - *GetNode Request* - est la requête à exécuter (elle utilise la variable initialiser à la ligne deux). *Constant timer* permet d'attendre un temps avant de recommencer. *Simple Data Writer* va écrire des données dans un fichier qui pourront être analysées.

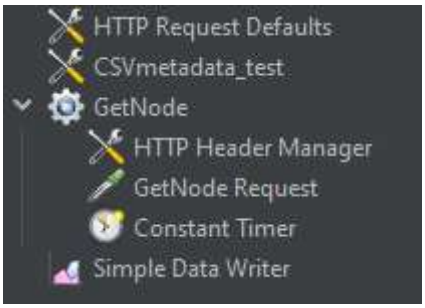


Figure 9 : Exemple de scenario dans JMeter

2.4 Mise en place et analyse des tests

Une fois que j'ai eu une bonne compréhension du fonctionnement de JMeter et après avoir réalisé quelques tests afin de vérifier qu'il avait tous les outils nécessaires, j'ai pu mettre en place des scénarios afin de tester des cas d'utilisation. Ces différents cas sont :

- Différente variante d'accès aux métadonnées d'un fichier avec 200 utilisateurs avec un temps d'attente entre chaque requête qui varie d'un scénario à l'autre.
- Une augmentation progressive du nombre d'utilisateurs avec une cadence rapide.

Les scénarios pour la modification de métadonnées et le téléchargement sont semblables à ceux-ci-dessus. Mais pour le téléchargement le nombre d'utilisateurs est beaucoup plus faible car il

a été découvert, grâce à certains tests réalisés au préalable durant ce stage, que le service subit des problèmes importants à partir de 8 utilisateurs simultanés (cf 2.4.2). Un dernier scénario mixte est présent, il permet de simuler une utilisation plus banale, il correspond à 50 utilisateurs qui envoient des requêtes aléatoires entre accès, modification des métadonnées et téléchargement.

L'image suivante montre le déroulement du script d'exécution des tests :

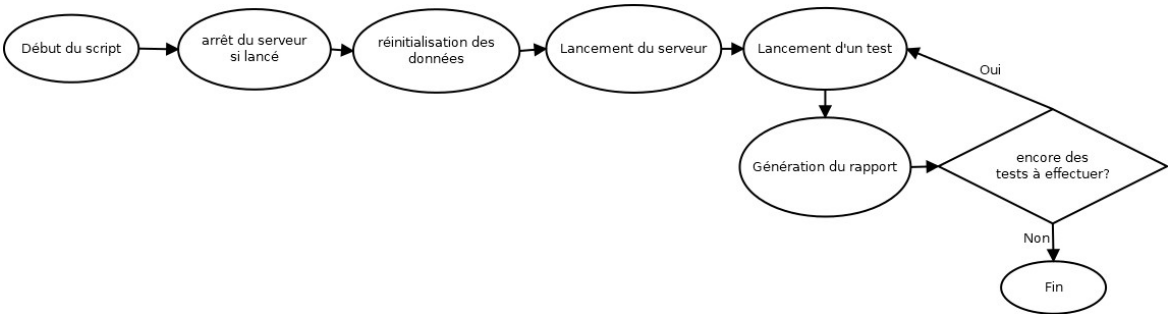


Figure 10 : Déroulement du script lançant les tests

La première étape est d'arrêter le service sur le serveur si le service est en cours d'exécution afin de pouvoir réinitialiser les données. En effet, le service doit être stoppé afin que la réinitialisation se déroule correctement. Ensuite le serveur est démarré. Une fois le serveur à nouveau opérationnel, le premier test est lancé sur la durée qui a été donné pendant la réalisation du scénario sur JMeter en mode interface. Quand le test est terminé, le rapport correspondant est généré, puis le test suivant est exécuté jusqu'à ce qu'il n'y en ait plus.

Avant de lancer ce script il faut remplir un fichier de configuration qui permet de mettre en place les différents chemins vers les ressources nécessaires. Il faut aussi lancer un script qui prépare en fonction des configurations données les scénarios afin de les adapter à différents ordinateurs.

Un autre script est disponible pour fusionner des rapports entre eux afin de mieux les comparer et qu'ils soient superposés sur les mêmes graphiques.

2.4.1 Requête sur les métadonnées

Lors des premiers tests sur le serveur nous avons remarqué que le temps de réponse semblait élevé et qu'un pic de temps de réponse avait lieux toutes les minutes environs. Après demande au service informatique, ces lenteurs étaient dû au fait qu'un analyseur de paquets* était actif sur le réseau du serveur. Après l'avoir désactivé, les temps de réponse ont été grandement améliorés et le pic n'était plus présent ou pour le moins grandement atténué. Ci-dessous le graphique illustre le temps de réponse élevé ainsi que les pics réguliers.

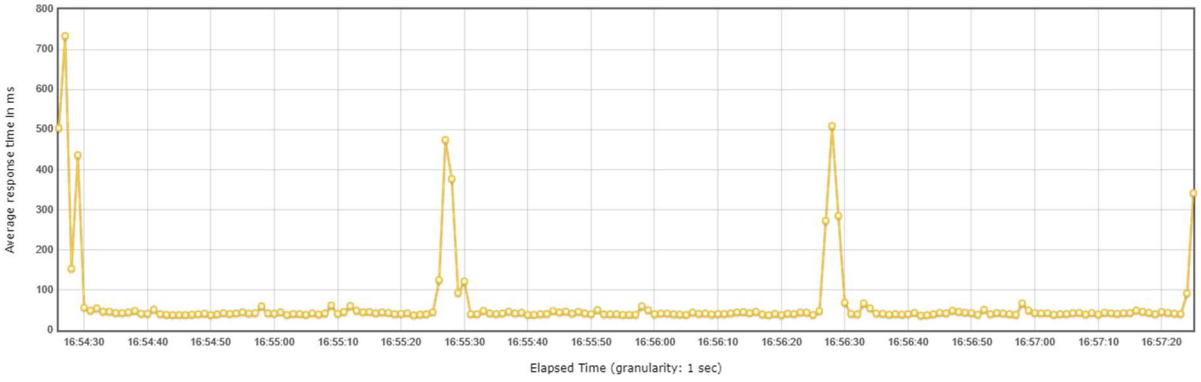


Figure 11 : Temps de réponse en fonction du temps avec des requêtes toutes les secondes

Le scénario, permettant de tester les limites, a montré un seuil d'utilisateurs (environ 4000 utilisateurs faisant des requêtes toutes les millisecondes) à partir duquel, de nombreuses requêtes ont une erreur en réponse. Le moment où ces erreurs commencent à se produire est visible sur les deux graphiques ci-dessous. Sur le premier, on observe que le temps de réponse tombe fortement vers 14:55:00 ; c'est à ce moment-là que les réponses sont des erreurs. De même sur le deuxième graphique à 00:00:49, l'utilisation du processeur diminue et ne remonte pas aussi haut qu'avant le pic descendant. Deux causes sont possibles :

- Soit le serveur n'arrive pas à suivre la cadence et envoie donc des erreurs.
- Soit c'est la machine qui lance les requêtes est surchargée et ne peut plus gérer en même temps autant d'envois de requêtes et réceptions de réponses.

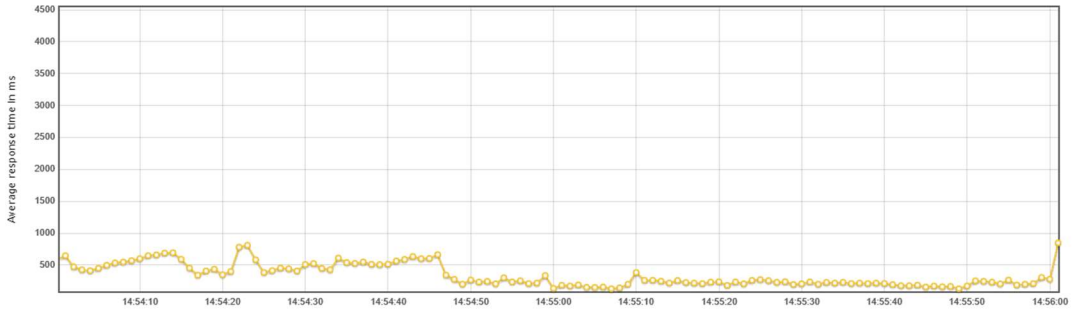


Figure 12 : Temps de réponse en fonction du temps pour le test des limites sur les métadonnées

Figure 13 : Utilisation CPU et mémoire pour le test des limites sur des requêtes sur les métadonnées

Après avoir analysé les résultats, il est plus probable que les erreurs viennent de la machine qui faisait les requêtes. En effet, l'utilisation du processeur côté serveur était élevée mais ne semblait pas en surcharge alors que l'utilisation du processeur côté client (machine envoyant les requêtes) était bien au-dessus de 90%.

Le graphique suivant compare le temps de réponse entre des requêtes sur des métadonnées associées à un fichier simple ainsi qu'à un dossier contenant une centaine de fichiers. On peut observer que le temps de réponse est légèrement plus élevé pour le dossier, ce qui semble évident car il faut lister tous ses fils. Donc plus un dossier a de sous fichiers plus le temps de réponse sera élevé. Finalement le temps de réponse reste acceptable dans les deux cas, moins de 100 ms dans les deux cas.

Un autre événement qui sera visible aussi sur d'autres graphiques est le temps de réponse au début des tests qui diminue progressivement. Une hypothèse qui pourrait expliquer ce phénomène est que le service est redémarré au début de chaque nouveau test (pour que le service soit dans un état initial) puis il y a une attente imposée par la phase de redémarrage avant pouvoir lancer le premier test. Il y a donc peut être un phénomène de cache* nécessitant d'attendre les premières requêtes sur la base de données pour que les performances soit au maximum.

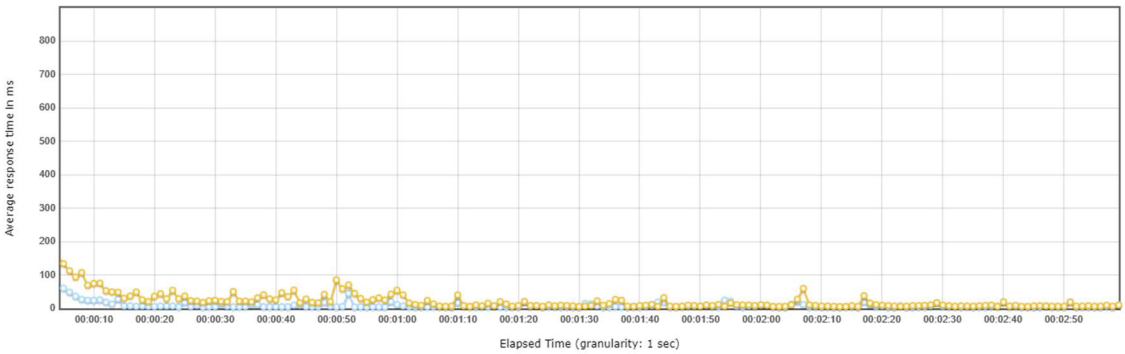


Figure 14 : Temps de réponse moyen en fonction du temps pour des métadonnées sur un fichier et sur un dossier

Enfin les dernières courbes ci-dessous comparent le temps de réponse entre des requêtes envoyées à des intervalles de temps différents. On observe le même phénomène que précédemment avec le pic au démarrage des tests. Chaque courbe correspond à un intervalle de temps différent d'envoi de requête :

- Courbe bleu 1ms
- Courbe verte 10ms
- Courbe jaune 100ms
- Courbe rouge 1s

On aperçoit que le temps de réponse n'augmente qu'assez peu en augmentant la fréquence des requêtes. Même avec des fréquences très élevé (1ms) il reste en dessous de 100ms.

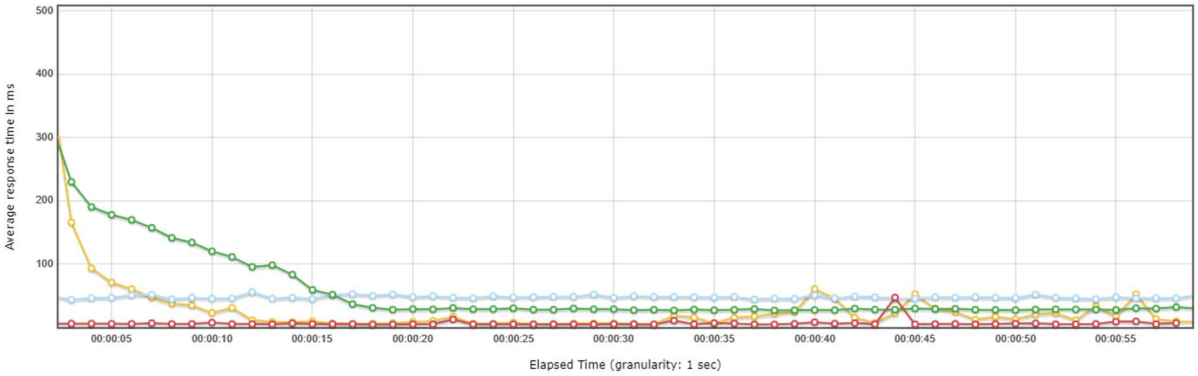


Figure 15 : Temps de réponse moyen en fonction du temps pour des requêtes sur les métadonnées sur un fichier toutes les : 1ms, 10ms, 100ms, 1s

Pour les requêtes de modification des métadonnées, les résultats sont très semblables à ceux d'accès au métadonnées (cf. ci-dessus). Le temps de réponse est également du même ordre. Son comportement est le même lorsque les requêtes sont faites sur des dossiers de grande taille. En effet, la modification de métadonnées effectue aussi un accès pour pouvoir renvoyer les nouvelles métadonnées du fichier ou dossier. Dans le cas d'un dossier la liste de ses fils est également envoyée.

2.4.2 Requête sur les fichiers

Les requêtes sur les fichiers concernent le téléchargement vers le client ou vers le serveur. Ces requêtes se déroulent en plusieurs étapes et sont donc plus complexes que les requêtes sur les métadonnées. Les premières étapes sont simples et ne font que demander au serveur de pouvoir réaliser le téléchargement, leur temps de réponse est donc très faible. L'étape la plus

intéressante est celle du téléchargement en lui-même. Le graphique ci-dessous montre le temps de réponse pour le téléchargement :

- La courbe orange représente le cas de 7 utilisateurs qui font des requêtes toutes les secondes.
- La courbe jaune représente le cas d'un utilisateur faisant des requêtes toutes les secondes.
- La courbe violette représente le cas d'un utilisateur faisant des requêtes toutes les 100 ms

On remarque bien que le nombre d'utilisateurs influence bien plus le temps de réponse que la fréquence des requêtes pour un seul utilisateur. On peut également remarquer que le téléchargement prend bien plus de temps que pour des requêtes sur les métadonnées. Cela est dû au fait que pour le téléchargement il faut lire le fichier sur le disque alors que les métadonnées sont stockées dans une base de données qui est de nature plus rapide. 7 est le nombre maximum d'utilisateurs avant que le service ne subisse d'importantes dégradation de ses performances. En effet, au-dessus de ce nombre le service bloque rapidement et ne peut plus répondre à aucune requête. Il est fort probable que cela soit causé par un problème de concurrence d'accès : une même liste est modifiée lors de l'initialisation et du téléchargement. Or s'il y a plusieurs requêtes en même temps, il est possible qu'il y ait des accès et modification de cette liste en même temps ce qui engendre une liste qui n'est plus synchronisée avec le reste

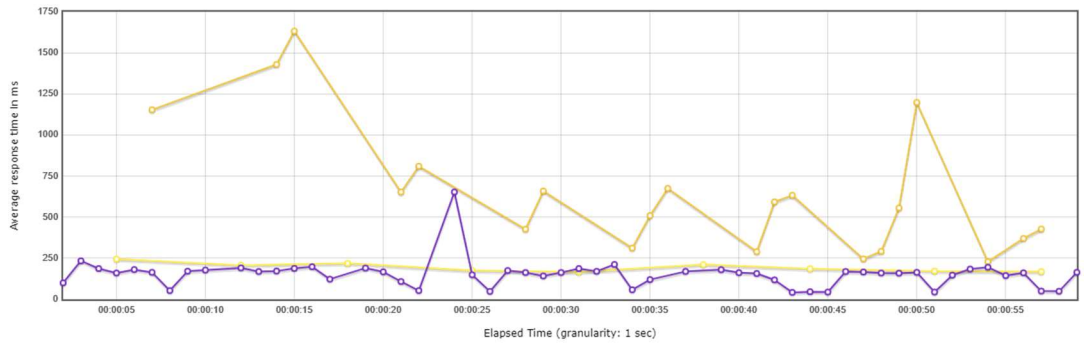


Figure 16 : Temps de réponse moyen en fonction du temps pour des requêtes sur les fichiers avec un utilisateur faisant des requêtes toutes les secondes, un autre toutes les 100ms et 7 utilisateurs faisant des requêtes toutes les secondes

du service.

Pour le scenario mixte (scénario mélangeant de accès/modifications de métadonnées et des téléchargements en simultanées), on observe que le téléchargement ne pose pas de problème dans le cas où tous les utilisateurs ne font pas forcément des actions identiques en même temps. Ce qui permet de conforter l'hypothèse d'un problème de concurrence.

Dans le graphique suivant, on observe que le temps de réponse d'un téléchargement de fichier (courbe verte) est bien au-dessus de tous les autres types de requête (accès/modification de métadonnée, requête intermédiaire du téléchargement). Il y a aussi sur cette courbe un pic assez marquant qui pourrait correspondre au démarrage de nouveaux téléchargements dans le cas où il y a plusieurs téléchargements en même temps. On remarque que les premières requêtes de type téléchargement ne démarrent que plus tard car il y a une suite de requête qui prépare le téléchargement qui prennent un certain temps.

Finalement, il y a également le même phénomène que précédemment lors des premières requêtes, juste après le démarrage du serveur.

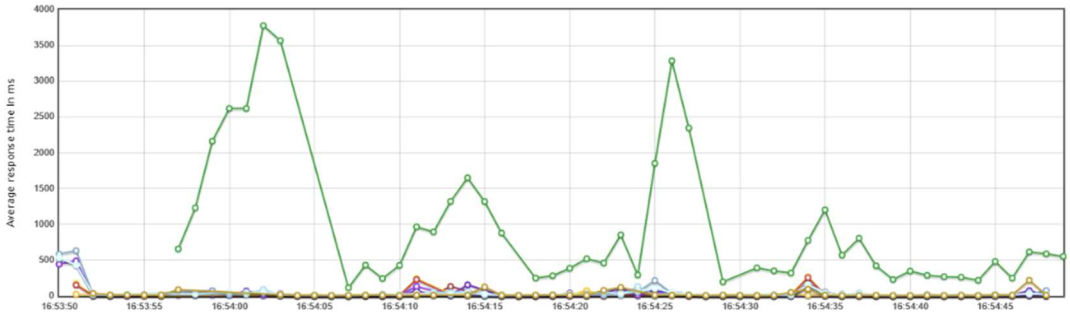


Figure 17 : temps de réponse moyen en fonction du temps pour un scenario mixte

Avec le graphique suivant représentant les ressources utilisées par le serveur, on remarque que lors du démarrage du test, il y a un pic d'utilisation du CPU dont l'origine n'est pas identifiée mais qui pourrait expliquer le temps de réponse élevé sur des requêtes simples au début du scénario. De plus, on remarque que l'utilisation mémoire reste constante, ce qui signifie que toutes les requêtes, qu'elles soient sur les fichiers ou sur les métadonnées, n'impactent que très peu la mémoire mais d'avantage le processeur ; le service reposerait alors plus sur du calcul que sur de la récupération de données.

Figure 18 : Utilisation du processeur et de la mémoire en % en fonction du temps pour un scenario mixte

Conclusion

La mise en place de ces tests a permis de démontrer que le prototype actuel à des performances qui sont acceptables en temps de réponse même pour des charges élevées mais que dans certains cas il n'est pas résilient et finit par ne plus répondre ou s'arrêter soudainement. Ces tests ont également montré quelles sont les requêtes les plus lentes et celles qui demandent le plus de ressources. Lorsque le nombre de téléchargements en simultané est supérieur à 7 le service ne répond plus. Cette limite est évidemment trop faible. Ce problème est dû à un accès concurrent. Cette suite de tests sous forme de scénario a été automatisée afin de pouvoir la configurer et la lancer facilement sur d'autres VOSpace. En effet, s'agissant d'un standard, toutes les requêtes et réponses suivent exactement les mêmes règles de construction et la même forme, permettant ainsi de pouvoir appliquer les mêmes tests quel que soit l'implémentation du standard. J'ai ainsi appris l'utilisation de nouveaux outils tel que JMeter et également approfondis mes connaissances en programmation de script Bash*. Des perspectives d'amélioration sont possibles, notamment la possibilité de changer la durée sur laquelle les tests sont réalisés. En effet dans le système actuel, les tests sont effectués sur des durées assez courtes, (de l'ordre de la minute) mais en général ces tests sont réalisés sur parfois plusieurs heures. Ajouter d'autres scenarios dans la suite de tests est une autre possibilité : toutes les fonctionnalités du VOSpace ne sont pas encore testées par la suite. Enfin, une autre piste d'amélioration serait de pouvoir réaliser les tests de manière distribuée c'est-à-dire plusieurs ordinateurs qui font les requêtes en même temps au lieu d'un seul. En plus d'en avoir appris davantage sur comment tester les performances, j'ai pu découvrir encore plus le monde de l'entreprise et plus précisément le travail dans un institut de recherche. J'ai également eu l'occasion de discuter avec mon tuteur de stage des idées, des avancées mais aussi des problèmes rencontrés via des réunions en visio-conférence ou en présentiel. La gestion du télétravail a été un autre aspect avec lequel j'ai dû me familiariser, puis plus tard dans le stage, la gestion distanciel présentiel.

Sitographie

Site de l'IVOA qui conçoit les standards de l'Observatoire Virtuel dont VOSpace *:*
http://ivoa.net/

Site de JMeter contenant sa documentation ainsi que les liens de téléchargement :
https://jmeter.apache.org/

Site de Katalon Studio contenant sa documentation ainsi que les liens de téléchargement :
https://www.katalon.com/

Site contenant la documentation liée aux plugins ainsi que les *plugins* réalisés par la communauté :
https://jmeter-plugins.org

Site de *Postman* contenant sa documentation ainsi que les liens de téléchargement :
https://www.postman.com/

Site de SOAPU*I* contenant sa documentation ainsi que les liens de téléchargement :
https://www.soapui.org/

Auteurs multiples, Test de performance, 2021
https://fr.wikipedia.org/wiki/Test_de_performance

Matthew Graham, Dave Morris, Guy Rixon, Pat Dowler, André Schaaff, Doug Tody, Brian Major, VOSpace, Version 2.1, IVOA Recommendation 20180620, 2018
http://www.ivoa.net/documents/VOSpace/20180620/REC-VOSpace-2.1.pdf

Glossaire

Analyseur de paquets : C'est un logiciel qui permet de capturer et d'analyser les données transitant sur un réseau informatique. Dans une entreprise cela est utilisé pour vérifier ce qui est fait sur le réseau et éviter des problèmes de sécurité par exemple.

Asynchrone : Une requête asynchrone est une requête qui se déroule sans que le client ne maintienne de connexion au serveur.

Bash : C'est un interpréteur de ligne de commande. Il permet de manipuler les fichiers d'un ordinateur via des lignes de code.

Cache : Le cache est une mémoire temporaire qui est plus rapide que la mémoire du disque d'un ordinateur. Cela permet à partir des premiers accès à des données d'accélérer les accès suivants.

CI/CD : Correspond respectivement à de l'Intégration Continue et du Développement Continu, c'est-à-dire qu'à chaque mise en production, il est possible d'effectuer des tests prédéfinis afin de vérifier que chaque nouvelle version d'un service et éventuellement un déploiement sur un serveur respecte les demandes.

Concurrence : C'est un problème qui a lieu lorsque différents processus d'un ordinateur utilisent une ressource et qu'au moins un d'eux la modifie. La conséquence est qu'en fonction de l'ordre des actions des processus le résultat sera différent et il n'est pas possible de prédire le comportement du système. Cela peut donc causer un disfonctionnement ou un blocage d'un système.

Csv : C'est un format de fichier structuré, il y a un entête et chaque information est séparé par une virgule. Par exemple :

Entête1,entête2,entête3
donnée11,donnée12,donnée13
donnée21,donnée22,donnée23
…

Débogage : Recherche et résolution de bug dans un système ou programme.

Interface : Dispositif permettant l'interaction entre différents acteurs. L'interface donne une manière unique d'accéder à certaines ressources. Ainsi aucun des acteurs n'a à faire attention à ce qu'il se passe de l'autre côté de l'interface utilisée.

IVOA : *International Virtual Observatory Alliance*, un consortium qui met en place des standards afin d'avoir une interopérabilité entre les services.

Le site de l'IVOA : http://ivoa.net/

JMeter : C'est un logiciel libre permettant de réaliser des tests de performance

Le site de JMeter : https://jmeter.apache.org/

Métadonnées : Informations associées à une donnée. Par exemple : une description, un sujet, …

Nœud : C'est le modèle de donnée de base au sein de VOSpace. Il peut s'agir d'un lien, d'un fichier ou encore d'un dossier.

Nom de domaine : C'est un identifiant de domaine sur internet, un domaine pouvant être n'importe quelle machine ou ordinateur relié à internet. Un nom de domaine est par exemple : google.fr

Plugin : Module permettant d'ajouter de nouvelles fonctionnalités à un outil ou programme.

Port : Les ports sont les portes d'entrées et de sorties d'un ordinateur, c'est par là que transitent les données venant de l'extérieur (internet par exemple) et aussi que sortent les données que l'ordinateur envoie.

Protocole : Un protocole spécifie des règles de communication entre un client et un serveur.

Requête : C'est une demande d'information d'un système à un autre. Le système serveur répondra avec une certaine donnée en lien avec la demande. Cette donnée peut être une image ou des données formatées en XML ou JSON.

Scenario : Un scenario est une d'action prédéfinie qui auront lieu lors d'un lancement de test.

Script : C'est un petit programme informatique qui exécute des commandes souvent sur des fichiers ou dossiers soit en les modifiant, soit en les créant à partir d'informations données. Cela permet d'automatiser des tâches répétitives.

Synchrone : Une requête synchrone est une requête qui se déroule alors que le client est connecté au service tout au long de son exécution.

Test de performance : Il a pour but de tester les performances d'un service en termes de temps de réponse sous certaines contraintes. Cela peut être des contraintes de nombre d'utilisateurs simultanés ou nombre de requêtes par seconde.

URI : *Uniform Resource Identifier*. Il s'agit d'une chaîne de caractères permettant d'identifier de manière unique une ressource. C'est de cette manière que VOSpace a choisi d'identifier un nœud.

UWS : L'*Universal Work Service* est un standard de l'IVOA qui spécifie comment exécuter des tâches asynchrones. Dans VOSpace, il est utilisé pour gérer la négociation de certaines requêtes (ex : téléchargement de fichier).

Document du standard UWS :
http://www.ivoa.net/documents/UWS/20101010/REC-UWS-1.0-20101010.pdf

VOSpace : Standard de l'IVOA spécifiant comment accéder et manipuler des fichiers.

Le document du standard :
http://www.ivoa.net/documents/VOSpace/20180620/REC-VOSpace-2.1.pdf

XML : L'*Extensible Markup Language* est un format de données qui stocke l'information de manière structurée à l'aide de balises. Voici un exemple de fichier XML :

```
<node xmlns="http://www.ivoa.net/xml/VOSpace/v2.0" version="2.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:type="vos:LinkNode" uri="vos://cds-vospace.com!vospace/[path]">
    <target>vos://cds-vospace.com!vospace/[path]</target>
    <properties>
        <property uri="ivo://ivoa.net/vospace/core#description">description</property>
    </properties>
</node>
```

On peut y observer la hiérarchie : *node* a pour fils *target* et *properties*. *Properties* a pour fils *property*. Le balisage est visible. Des schémas peuvent être précisés afin de pouvoir vérifier la forme du XML.