Outil de modélisation de performances des migrations de données inter-cloud

Antoine Martin - Carole Bonfré

Février 2015

Résumé : Ce projet a pour but de permettre de choisir, selon une localisation géographique et des paramètres donnés, la meilleure solution d'hébergement *Cloud* existante. De nombreux chercheurs rencontrent le besoin de déplacer des Téraoctets de données, cet outil leur permettra donc d'optimiser la migration de leurs données. À terme, ce projet Open Source proposera une solution unique sur le marché.

I Introduction

Dans un monde toujours plus interconnecté, nos données sont de plus en plus dématérialisées et éparpillées. Nous sommes amenés à les transférer d'un hébergeur à un autre et dans le but d'optimiser ces opérations, il nous a été demandé de nous poser différentes questions concernant l'évaluation des méthodes de transfert.
Cette thématique s'inscrit dans le cadre de l'équipe de recherche Avalon du LIP de l'ENS de Lyon, qui propose des solutions pour la distribution des calculs dans des fédérations de *Cloud*. La distribution de ces calculs implique de nombreux mouvements de données inter-cloud. Pour contribuer à l'amélioration des travaux de recherche des membres de l'équipe Avalon, nous avons été amenés à étudier, proposer, et développer un outil permettant d'évaluer la performance de transferts de données entre plusieurs hébergeurs de "Cloud". Cet outil permettra soit d'effectuer des tests de performances « à la demande », soit de récupérer les résultats déjà obtenus lors de précédents tests.

Ce projet, nous l'espérons, pourra non seulement aider les chercheurs de l'équipe Avalon de l'ENS mais également offrir un outil à tous projets ou personnes ayant besoin d'évaluer les temps de transfert entre solutions *Cloud* (académiques comme privées).

II Recherche et Analyse

A Étude de l'existant

Cette étude peut être divisée en deux parties. Tout d'abords, nous avons détaillé les offres de solution de stockage “Cloud” disponibles sur le marché afin de pouvoir d'en dresser les caractéristiques en terme de prix et de performance. Nous avons ensuite cherché si il existait déjà un outil capable de réaliser des tests de performances sur des “Cloud”, l'abscence de résultats pouvant justifier le développement de notre propre application.

1 Solutions de stockage *Cloud*

TABLE 1 – Tableau comparatif des "*Cloud*"

drive	API	Emplacement	Libre	Espace de stockage	Limitation	SDK
Dropbox	Oui	S3	Gratuit / Propriétaire	2Go	Oui(N/A)	Oui
Google drive	Oui	lien	Gratuit / Propriétaire	15Go	10 000 requêtes /jour, 10 requêtes /sec/user	Oui
S3	Oui	lien	Gratuit / Propriétaire	5Go	20 000 GET, 2 000 PUT / mois	Oui
Onedrive	Oui	?	Gratuit / Propriétaire	15Go	Oui (NA)	Oui
Cloud Orange	Oui	Paris (Sénégal ?)	Gratuit / Propriétaire	10Go à 100Go	Oui (2Go par fichier)	Oui
Hubic	Oui	France (Paris, Roubaix)	Gratuit / Propriétaire	25Go	Oui (10Go par fichier)	Oui
Microsoft Azure	Oui	lien	Gratuit / Propriétaire	100To	Oui (NA)	Oui
iCloud	Oui	USA (Caroline du Nord)	Gratuit / Propriétaire	5Go	Oui (15Go par fichier)	Oui
Google Cloud Storage	Oui	lien	Payant / Propriétaire	1To	Oui 5Tb par fichier	Oui
Cloud bouygues	Non	USA (Pogoplug)	Payant / Propriétaire	5Gb	Oui (NA)	Non
SFR Cloud	Non	Paris	Payant / Propriétaire	100Go	Oui (NA)	Non

*La mention Oui(N/A) pour la colonne des limitations signifie une présence de limitation non explicitée par l'hébergeur.

Nous avons remarqué lors de notre analyse que l'hébergeur Dropbox utilise en fait les services d'Amazon S3. Dropbox étant un des services les plus populaires, nous avons décidé de le choisir : il apporte probablement des services supplémentaires. Nous pensons qu'il est également intéressant de tester les différences de performance entre Amazon S3 et Dropbox. Pour les mêmes raisons, il aurait également été intéressant de pouvoir vérifier les différences entre Google Drive et Google Cloud Storage.

Nous avons aussi étudié la possibilité de sélectionner OwnCloud, solution libre pour mettre en place un "*Cloud*" privé, parmi les hébergeurs (mais l'utilisateur doit posséder une machine avec OwnCloud installé, ce qui signifie la possession d'une machine serveur où Owncloud fonctionnerait contrairement aux autres hébergeurs qui eux ne nécessite pas de machine serveur). Il ne possède pas de limitations particulières, l'espace de stockage est "infini" (il dépend du serveur et de la configuration de Owncloud), et nous avons trouvé un *SDK* développé par un tiers qui semble exploitable pour notre application. À terme, l'ajout de OwnCloud peut donc être envisagé.
Après analyse des différents acteurs du marché de stockage en ligne, nous avons décidé de sélectionner les trois hébergeurs suivant : Dropbox, Amazon S3 et Google Drive car ce sont les plus populaires en ce moment et qu'ils possèdent tous les trois des *SDK* qui facilitent le développement de notre outil. En revanche, les espaces de stockage

sont parfois assez limités.

2 Solutions d'évaluation de performance des stockages *Cloud*

Trois projets ont été identifiés durant cette étude. Le premier, HP Performance, est une solution propriétaire payante. Parmi de nombreux outils, elle propose de se placer dans une zone géographique pour provisionner un générateur de charges (simulation d'une utilisation intense d'un "Cloud"). Le second projet, COSBench, est une solution libre mais plutôt limitée puisqu'elle ne concerne que Swift Storage et Amazon S3. Elle permet d'exécuter des tâches sur des outils distants et de les surveiller. Ces tâches peuvent être du test de performance de débits ou encore des tests de charge. Enfin, Cloudcreener est une solution payante que nous n'avons pas pu tester. Il semblerait qu'il réalise le genre d'opération que KYD est censé faire d'après leur site internet, cependant cet outil n'étant pas libre il ne remplit pas toutes les conditions de notre projet.

Nous les avons comparés à notre projet baptisé KYD (Know Your Data).

TABLE 2 – Tableau comparatif des solutions trouvées

	HP Performance	**CosBench**	**Cloudcreener**	**Kyd**
Générique	Oui	Oui	Non	Oui
Open source	Non	Oui	Non	Oui
Modulaire	Non	Non	Probablement	Oui
Interface graphique	Oui	Oui	Oui	Non
Limites	Propriétaire	Swift Storage et S3	Propriétaire, tests spécifiques	limites des *Cloud*
Stockage des résultats	Exports multiples	Exports multiples	Web	Base de données

Un autre outil en cours d'implémentation a aussi attiré notre attention : PerfKit. Développé par Google, il est très similaire à notre projet mais comporte des différences majeures puisqu'il effectue ses tests seulement pour des machines virtuelles. Il nécessite de pouvoir installer des logiciels sur les serveurs des *Cloud*, ce qui n'est pas réalisable puisqu'il faut avoir l'accord des hébergeurs (le projet se limite donc aux propres hébergeurs de Google et, actuellement, à Microsoft Azure et Amazon AWS qui sont des serveurs virtuels privés). Ce projet ne peut pas atteindre les hébergeurs que nous ciblons et ne constitue donc pas un concurrent à proprement parler. On peut aussi noter que nous avons démarré nos travaux de recherche le 19 Janvier 2015 et que cet outil est devenu public le 11 Février 2015. On peut donc remarquer que l'application que nous avons développée se situe sur un secteur qui n'est pas encore exploité, ce qui veut dire que nous proposons une solution unique.

B Définition de la structure et modélisation

Un grand nombre de paramètre est à prendre en compte pour obtenir des résultats cohérents et réutilisables. Il faut connaître la taille d'un fichier transféré ou le fichier lui-même, l'emplacement géographique de l'utilisateur (de bout en bout), la date du test, les hébergeurs à tester, ainsi que le type de transfert (*upload* ou *download*) pour sauvegarder les résultats avec précision. En retour, l'application propose une liste des différents hébergeurs testés avec les temps obtenus, le meilleur choix étant mis en valeur. Tous les résultats calculés sont stockés en base de données pour pouvoir être réutilisés lors de tests ultérieurs.

C Choix des technologies

Le fait de travailler avec beaucoup de paramètres différents impose de pouvoir tester toutes les solutions possibles de manière exhaustive. Il faut également pouvoir stocker ces dernières de la façon la plus adapté possible pour

pouvoir les retrouver facilement. Deux outils ont donc été retenus : *Execo Engine* et *MongoDB*.

1 *Execo Engine*

Execo est un outil développé par Matthieu Imbert, Laurent Pouilloux, Jonathan Rouzaud-Cornabas, Adrien Lèbre et Takahiro Hirofuchi qui sont des chercheurs affiliés au LIP (Laboratoire de l'Informatique du Parallélisme) de Lyon ayant de multiples fonctionnalités dans le but de réaliser des expériences reproductibles sur des système distribués, d'automatiser des tâches administrateur et de créer des expériences reproductibles. Dans notre cas, seule la partie *Engine* de *Execo* est intéressante puisqu'elle permet de réaliser le produit cartésiens des paramètres pour générer l'ensemble des configurations possibles. Par exemple, si l'utilisateur souhaite tester le transfert de fichiers de différentes tailles sur les différents hébergeurs, Execo Engine génère les combinaisons "taille1/drive1", "taille1/drive2", *etc.* Toute combinaison n'ayant pas pu être testée à cause d'une erreur est enregistrée et peut être relancée plus tard. Il s'agit donc d'un excellent outil de test qui chronomètre également tous les tests effectués de façon très précise.

2 *MongoDB*

Notre application possédant une structure de données appelée à être modifiée régulièrement, il fallait qu'elle possède un système de gestion de base de données adapté. Sachant qu'une seule table serait nécessaire mais qu'elle contiendrait, à terme, un très grand nombre de tuples, nous nous sommes tournés vers le système non relationnel qu'est *MongoDB*. Sa vitesse de traitement associée aux index permet de requêter très rapidement pour fournir un résultat à l'utilisateur.

III KYD outil de "*benchmark*"

KYD est une application implémentée en langage *Python*. Son développement se divise en deux parties : l'interaction avec les hébergeurs et l'interaction avec l'utilisateur. Nous avons donc, un système de *benchmark* et un système de requête dans la même *API*.Pour plus d'efficacité, nous avons implémenté ensemble toutes les parties sur *Dropbox* puis nous nous sommes partagés le travail sur *Amazon S3* et *Google Drive*.

A Interaction "*Cloud*"

Dans un premier temps, il nous a fallu mettre en place toutes les connexions avec les hébergeurs sélectionnés. Nous avons utilisé les *SDK* propres à chaque *Cloud* et nous avons ensuite effectué une série de tests unitaires pour vérifier le fonctionnement du transfert de fichiers en *download* et en *upload*.

Cette partie du code se veut très modulaire puisqu'elle doit permettre l'ajout d'autres hébergeurs. Pour faciliter ces ajouts, nous avons créé une classe par hébergeur (il suffit donc d'implémenter les fonctions désirée pour effectuer un ajout). Nous avons rencontré quelques difficultés durant cette étape puisqu'une telle implémentation demande de s'adapter à l'utilisation de chaque SDK. Malgré tout, les hébergeurs sélectionnés étant très populaires, ils disposent d'une large communauté qui permet de régler rapidement la majorité des problèmes.

B Interaction utilisateurs

Actuellement, l'interface de communication avec l'utilisateur se fait dans un terminal *Unix*. L'utilisateur demande à l'application la meilleure solution de transfert de données selon ses paramètres et, si la base de données MongoDB contient des informations suffisamment similaires (zone géographique et taille de fichier proches), le résultat est retourné à l'utilisateur. Dans le cas contraire, KYD demande à l'utilisateur s'il accepte d'effectuer un test selon certains paramètres donnés pour enrichir la base de données et pouvoir répondre lors d'une prochaine requête. Pour minimiser le nombre de paramètres à saisir pour l'utilisateur, sa localisation géographique est automatiquement détectée grâce à son adresse IP et à deux *API* de géolocalisation (ip-api et Telize qui se relaient si le serveur de l'une d'entre elles est hors service).

Le schéma ci-dessous synthétise le fonctionnement de notre application dont les divers éléments ont été expliqués précédemment. L'utilisateur envoie ses paramètres à KYD qui, dans un premier temps, le géolocalise avec *ip-api* ou *Telize*. Les paramètres sont ensuite utilisés pour requêter la base *MongoDB* et renvoyés une solution si elle existe. Dans le cas contraire, l'application demande à l'utilisateur s'il souhaite effectuer les tests. S'il accepte, Execo Engine

démarre les tests en contactant un à un les hébergeurs et, une fois terminé, les résultats sont enregistrés dans la base de données *MongoDB* et envoyés à l'utilisateur.

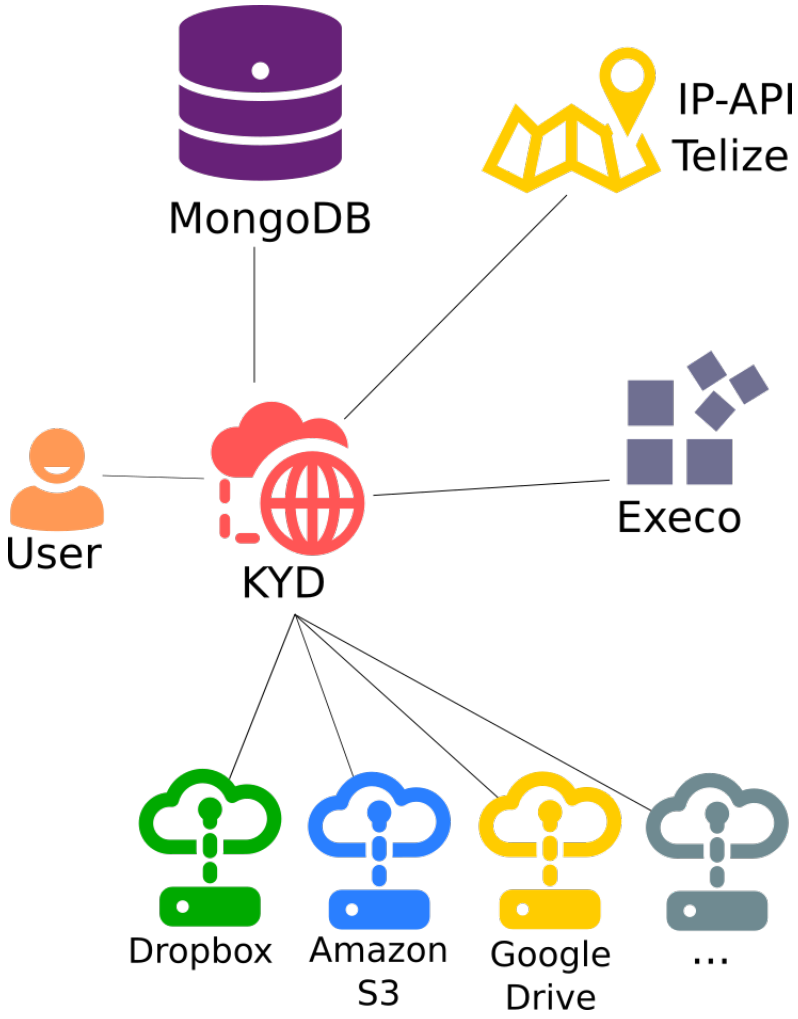

FIGURE 1 – Architecture de l'application

C Améliorations

À terme, plusieurs améliorations sont envisageables. Ajouter des paramètres pour augmenter la précision des tests ferait partie de ces améliorations. Il serait par exemple possible de vérifier l'efficacité des compte premium qui apportent peut être de meilleures performances. Il faudrait également compléter la base de données pour pouvoir répondre le plus souvent possible à l'utilisateur sans avoir à lui faire effectuer des tests parfois coûteux en temps. Il serait alors nécessaire d'effectuer des tests à plusieurs emplacements géographiques dans le monde.

Enfin, une interface Web plus conviviale pourrait rapporter les résultats à l'utilisateur à la place de la console. Cette amélioration se veut majoritairement esthétique mais pourrait apporter un certain nombre de renseignements supplémentaires à l'utilisateur. Notre application n'est pas encore destinée à des utilisateurs novices (utilisation d'un système *Unix* et uniquement en terminal), une interface web pourrait la rendre plus ergonomique. Enfin il serait intéressant que nos résultats, bien que stockés dans une base de données, soient plus facile d'accès.

IV Analyse des résultats

Suite à de nombreux tests, nous avons pu obtenir différentes courbes traduisant les divers aspects de notre projet. Nous désirions comparer les trois hébergeurs pour une localisation donnée, mais nous voulions aussi savoir s'il existait des différences selon le moment de la journée ou selon le jour de la semaine.

Pour ces tests, nous avons réalisé un minimum de 15 tests par "*Cloud*" et par tranche horaire (par exemple : 15 tests à 15h pour Amazon, GoogleDrive et Dropbox), les résultats ci-dessous représentent la moyenne de ces tests. Ils ont tous été réalisés avec les *SDK* de chaque "*Cloud*".

Pour les figures 1 et 2, les tests ont été réalisés depuis le campus de la Doua à Lyon 1. Quinze tests ont été effectués en *upload* et en *download* avec une taille de fichier différente puis la moyenne de ces tests a permis d'obtenir ces résultats.

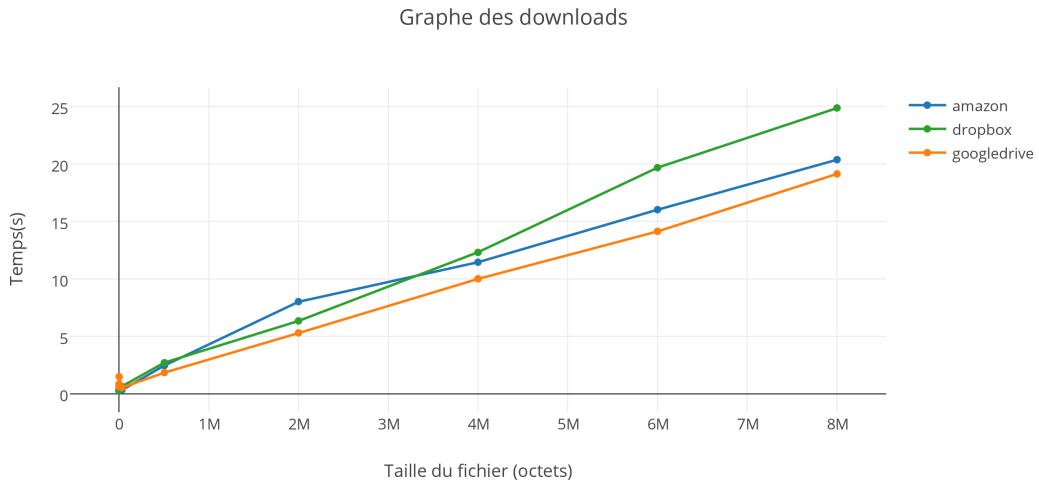

FIGURE 2 – Graphe des *downloads*

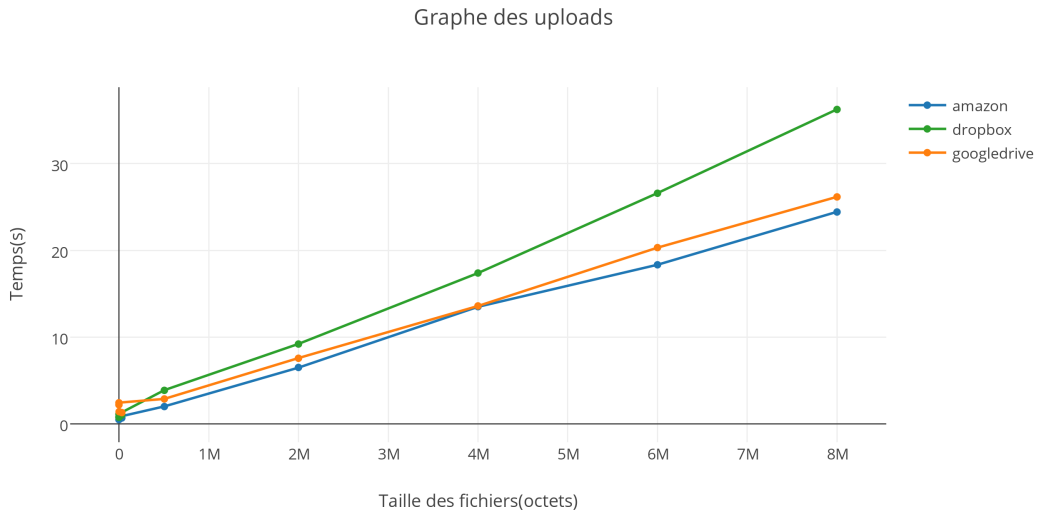

FIGURE 3 – Graphe des *uploads*

Il est clairement visible que, dans cette situation, Dropbox est le choix le moins pertinent sauf en ce qui concerne la connexion. Amazon et Google Drive seront donc des choix beaucoup plus logiques puisqu'on peut observer une différence d'environ 20 points entre Dropbox et ses deux concurrents en ce qui concerne les performances (et cette différence ne cesse de croître avec l'augmentation de la taille des fichiers).

Pour les deux figures suivantes, les tests ont été effectués depuis un serveur virtuel situé à Londres. Ils ont été lancés toutes les heures pendant vingt quatre heures un dimanche avec une taille de fichier de 10Mo à l'aide de la crontab du serveur.

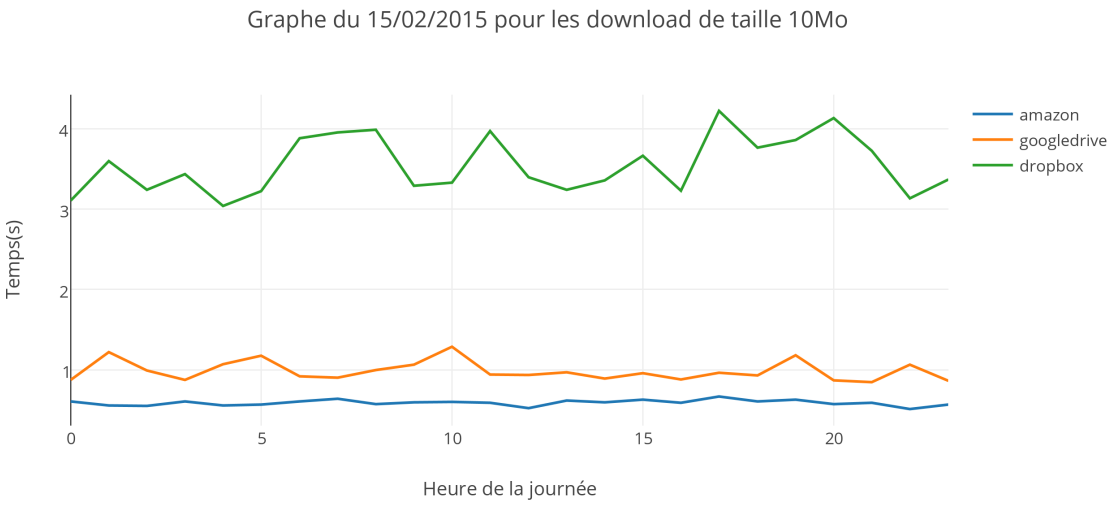

FIGURE 4 – Graphe des *downloads* du 15 de taille 10Mo

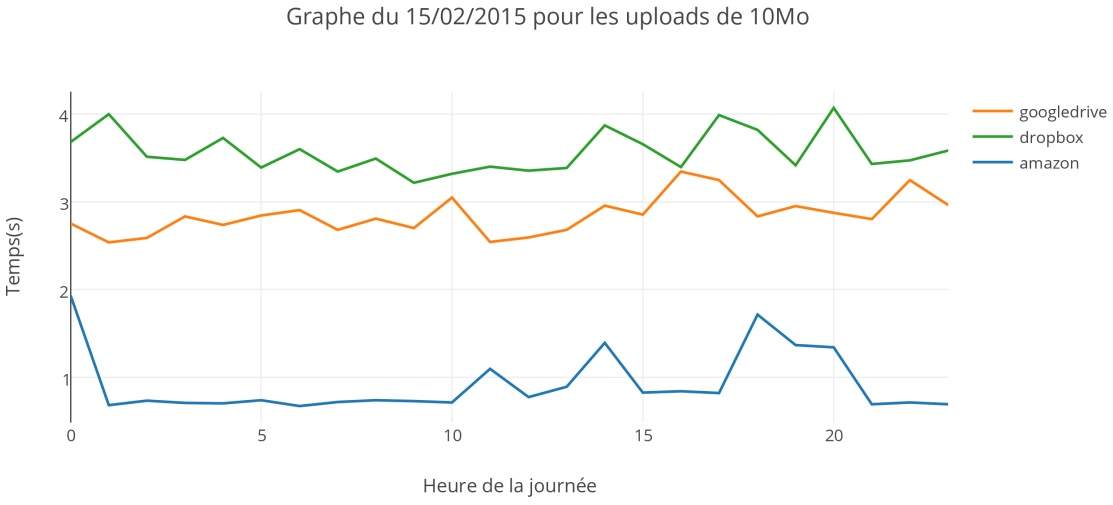

FIGURE 5 – Graphe des *uploads* du 15 de taille 10Mo

En "*upload*", on observe qu'Amazon suit un schéma connu qui comporte des pics de latence à 11h, 14h puis entre 18h et 20h. On suppose que cette situation est due au grand nombre de connexions simultanées durant une plage horaire plus bondée que les autres. Les performances de Dropbox sont, cette fois encore, très nettement inférieures à celles de ses concurrents.

Dans le même context que les deux derniers graphes situés ci-dessus, nous avons effectué les mêmes tests sur un autre jour de la semaine : le Mercredi.

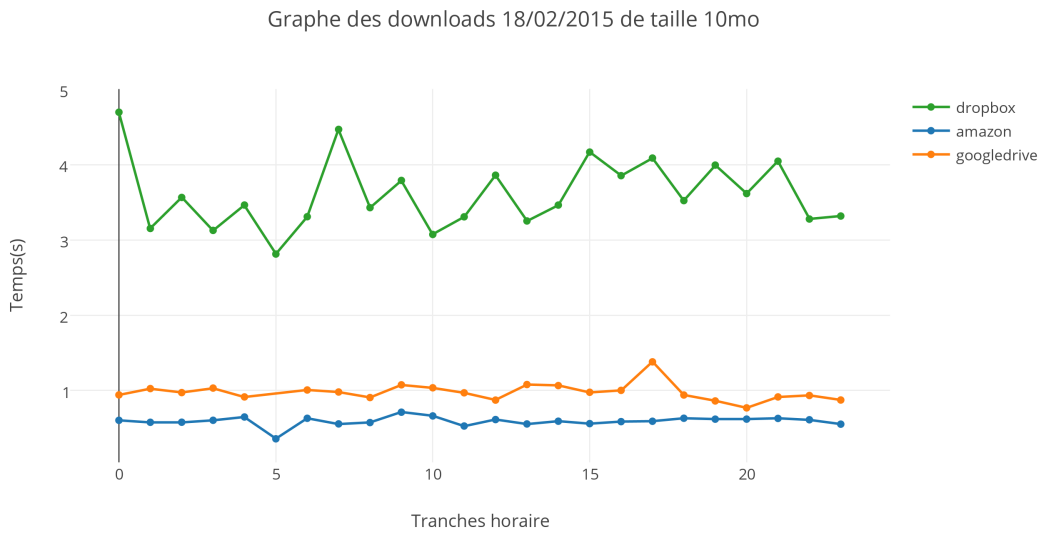

FIGURE 6 – Graphe des *downloads* du 15 de taille 10Mo

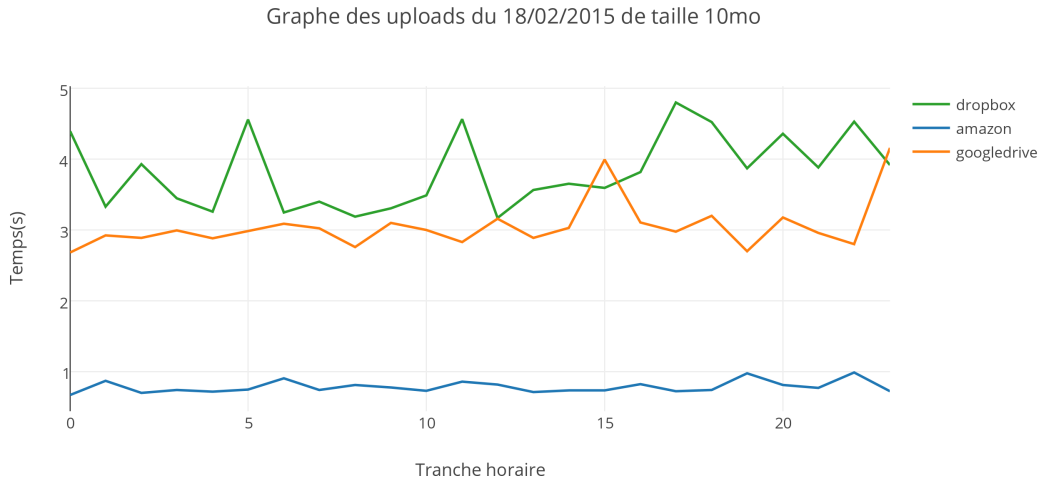

FIGURE 7 – Graphe des *uploads* du 15 de taille 10Mo

On remarque ici l'absence d'une donnée concernant le "*Cloud*" GoogleDrive dans la partie "*download*" car il était en maintenance à 5h du matin. Notre outil prend en compte ce genre d'erreur et permet de continuer les mesures sur les autres hébergeurs.

On peut observer que l'hébergeur *Amazon* et son service *S3* reste loin devant les autres "*Cloud*" en gardant ses moyennes en-dessous de la seconde. *Google Drive* possède de bonnes mesures concernant la partie "*download*" de son service, cependant on remarque qu'il rencontre quelques difficultés concernant la partie "*upload*". *Dropbox* reste bon dernier tout au long de ces tests, nous supposons que cela est probablement due à la surcouche que *Dropbox* ajoute par rapport à *S3*. En effet, comme nous vous l'avions expliqué plus tôt, le service *Dropbox* utilise le service d'*Amazon*.

V Conclusion

Le besoin d'optimisation des méthodes de transfert se faisant de plus en plus fort, il est devenu nécessaire de pouvoir sélectionner ses hébergeurs avec soin. Cette thématique ouvre un large champs de possibilités dans lequel notre stage a trouvé sa place.

Durant ce projet, nous avons donc analysé, modélisé et implementé un outil permettant d'estimer la meilleure solution de transfert de données entre plusieurs hébergeurs. Pour ce faire, une phase de recherche était nécessaire puisqu'il fallait vérifier qu'un tel outil n'existait pas déjà et, dans ce cas, sélectionner les meilleures solutions "*Cloud*".

Les résultats obtenus suite aux tests ont permis de vérifier la validité de notre application. En effet, un écart trop important entre les résultats aurait entraîné un manque d'homogénéité des moyennes. Dans une telle situation, KYD n'aurait pas pu fournir de résultat valable puisque les temps obtenus auraient été trop aléatoires et nos recherches auraient prouvées qu'un tel outil n'était pas réalisable.

Ces résultats sont donc très satisfaisants puisque le développement de KYD va pouvoir se poursuivre pour proposer des solutions plus précises sur plus d'hébergeurs. Cette application va très certainement étendre son champs d'action dans le monde de la recherche puisqu'une poursuite de projet sera normalement lancée sous peu pour améliorer KYD.

Ce projet est donc une excellente expérience pour nous puisqu'il nous amène sur un domaine encore très peu exploité et prometteur. Il nous a aussi donné la chance de pouvoir travailler dans l'univers de la recherche qui nous était jusqu'alors inconnu et de collaborer avec des chercheurs dans une ambiance dynamique. Nous avons donc pu faire beaucoup de découvertes au cours de ces quelques semaines et décider avec plus d'assurance de notre orientation future.

VI Annexes

Les outils utilisés :
- Github - Gestion de version
- Travis.cl - Intégration continue
- Ip-api et Telize - API de géolocalisation
- Execo - Gestionnaire de processus unix
- MongoDB - Base de données
- Dépendances python dans le fichier requirements.txt
- LaTeX
- landscape.io - mesures de qualité du code python
- Le café du LIP & du Nautibus

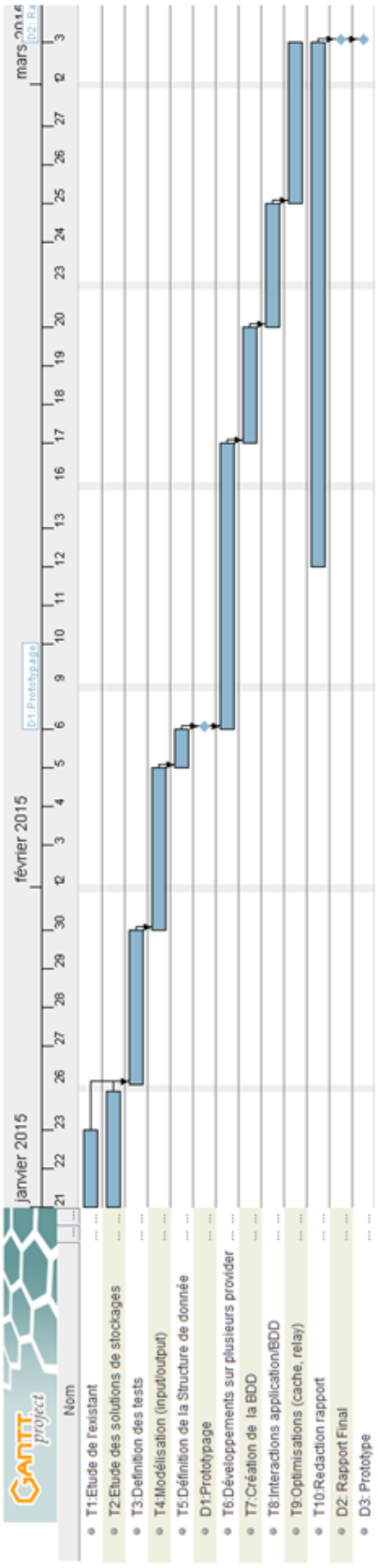

FIGURE 8 – Diagramme de Gantt