

Projet de Programmation Numérique :
Extension de la Résolution de Noms pour le
HPC

YEHOSSOU Kakpo, KONÉ Sirata, MLARAHHA Hassan

27 avril 2022

Table des matières

Introduction	4
1 Présentation générale	5
1.1 Contexte du projet	5
1.1.1 Le Standard PMIx	5
1.1.2 Le protocole DNS	6
1.2 Objectifs du projet	7
2 Implémentations	9
2.1 Fonctionnement du proxy DNS	9
2.1.1 Les structures de données et routines associées	9
2.1.2 La communication client/serveur	9
2.2 Principe de la table de hachage	10
2.3 Interface Clef-Valeur	11
2.3.1 Opération <i>Get</i>	11
2.3.2 Opération <i>Set</i>	11
2.3.3 Opération <i>Delete</i>	11
2.4 Système de <i>log</i>	12
2.4.1 Traitement du fichier	12
3 Choix d'implémentation et Mesure de Performance	13
3.1 système de cache	13
3.1.1 table de hachage	13
3.1.2 Sérialisation des données en échange	13
3.1.3 Test de validation	14
3.2 Parallélisation et Performance	14
3.2.1 Table de hachage	14
4 Organisation et déroulement	16
4.1 Structure du projet	16
5 répartition des tâches	16
6 Niveau d'achèvement	18
6.1 Limites du projet	18
6.2 Travaux Futurs	18
Conclusion	19

Références	20
Annexes	21
Table des figures	

Introduction

Dans le cadre du projet de programmation numérique, il nous a été demandé de réaliser un proxy DNS avec des fonctionnalités étendues, permettant de couvrir les besoins en résolution de noms pour des machines parallèles dédiées au calcul haute performance, sur une architecture distribuée.

Nous présenterons dans une première partie le projet de manière générale, à savoir le contexte et les objectifs à atteindre. Ensuite, nous expliquerons plus en détail la conception et les implémentations effectuées. Puis, nous expliquerons notre organisation et la répartition des tâches. Enfin, nous aborderons un point de vue critique sur l'ensemble de la solution élaborée, ainsi que les améliorations envisageables.

1 Présentation générale

Ce projet se présente sur plusieurs axes principaux à savoir :

1.1 Contexte du projet

Dans le domaine du calcul haute performance, la communication entre machines (ou des processus) est l'une des préoccupations les plus importantes en terme d'optimisation. En effet, les entités parallèles doivent être en mesure de pouvoir mener à bien les opérations complexes qui leur sont assignées, et ceci quelque soit la configuration de l'architecture (partagée ou distribuée) afin de résoudre le problème posé.

Pour s'échanger des données, les processus parallèles (jusqu'à plusieurs millions) doivent au préalable être capable de s'identifier et de se reconnaître les uns par rapport aux autres, afin de s'interconnecter, et ce de la manière la plus efficace possible. C'est en ce sens qu'un mécanisme de résolution de noms performant est essentiel pour l'identification de ces entités. Dans le cas des réseaux de machines distribuées, des solutions existent pour accomplir cette nécessité. Dans les sections suivantes, nous présentons deux de ces solutions : le PMIx et le DNS qui agissent souvent de manière complémentaire lors du lancement des applications parallèles.

1.1.1 Le Standard PMIx

L'interface PMIx (*Process Management for Exascale Environments*) est une API (*Application Programming Interface*) et plus récemment un standard qui fournit un accès portable et bien défini aux services communément utilisés par les systèmes de calcul distribué. Plus précisément, la PMIx implémente un stockage clef-valeur distribué permettant l'échange des informations de connection lors du lancement des programmes MPI. De plus, ce stockage est généralement géré par le batch manager, et donc présent avant que le programme ne soit lancé, agissant alors comme tiers de connectivité.

Cette interface a été développée et distribuée dans le cadre du projet MPICH (implémentation portable et *open-source* de la bibliothèque de calcul parallèle MPI). À l'origine, elle était utilisée comme moyen de communication (filaire) entre processus, et également comme un outil de déploiement de processus. L'API a connue plusieurs versions respectivement appelée PMI1, 2 et plus récemment X pour Exascale. Cette dernière fait l'objet d'un processus de standardisation cf. pmix.org.

Plus précisément, PMIx intervient lors du lancement des programmes MPI (et plus généralement distribués) pour :

- Donner le nombre de processus lancés (rank et size) ;
- Permettre l'échange des informations de connexion (point d'intérêt dans notre étude) ;
- S'interfacer avec le batch manager pour lancer des processus dynamique.

L'objet de notre étude sera d'étudier la convergence de DNS avec PMIx afin de déterminer s'il existe des symétries entre ces deux services, et donc un moyen de les coupler.

1.1.2 Le protocole DNS

Le protocole DNS (*Domain Name Service*) est un mécanisme qui permet d'obtenir l'adresse IP d'une machine à partir d'un nom de domaine (sous la forme d'une URL), ou le nom de domaine d'une machine à partir d'une adresse IP. Dans le second cas, on parle de résolution inverse. La conversion entre nom de domaine et adresse IP est possible grâce à la mise en place d'un espace de noms hiérarchisé. Ce dernier est organisé sous la forme d'un arbre à plusieurs niveaux, calqué sur la structure des organismes gouvernementaux et non-gouvernementaux. Le caractère "." est utilisé pour marquer la frontière entre deux niveaux hiérarchiques.

La création de ce protocole à une importance dans l'existence d'Internet tel que nous le connaissons à l'heure actuelle. En effet, en 1983, les protocoles TCP et IP ont été adoptés pour identifier les machines hôtes du réseau ARPANET (*Advanced Research Projects Agency Network*). La résolution de noms était réalisée par l'intermédiaire d'une table de correspondance entre noms d'hôtes et adresses IP. Cette table était enregistrée dans le fichier *hosts.txt* (ancêtre de */etc/hosts*, ce dernier étant maintenu par le NIC (*Network Information Center*)). La transmission du fichier était effectuée via le protocole FTP à tous les hôtes du réseau. C'était alors les débuts d'Internet.

Avec l'augmentation exponentielle du nombre de machines hôtes, la mise à jour du fichier *hosts.txt* devenait complexe (encombrement de la bande passante du NIC). De plus, la croissance d'Internet étant inévitable, les divers organismes impliqués ont exprimé leur besoin d'autonomie dans la gestion de leurs communications à travers ce réseau. Dans ce contexte, Paul Mockapetris et John Postel développent entre 1983 et 1984 une solution qui utilise des structures de base de données distribuée : les *Domain Name Systems* (RFCs 882 et 883, aujourd'hui obsolètes). Cette première approche aboutit en novembre 1987 à la spécification du protocole DNS (RFCs 1034 et 1035).

Ainsi, le DNS a été créé avant tout pour rendre le plus agréable possible la communication entre les machines du réseau Internet.

1.2 Objectifs du projet

Comme énoncé dans le préambule de cette partie, la principale problématique consiste à trouver un moyen efficace et scalable d'interconnecter des processus entre eux, par intermédiaire d'un système de résolution de noms. En d'autres termes nous souhaitons étudier une convergence possible entre la Process Management Interface (PMI) et le protocole DNS. En effet ils ont des rôles comparables, permettre à des machines de se retrouver, l'un est temporaire à un job donné l'autre est souvent un service allant du calculateur lui même (cas d'un réseau isolé) à l'Internet mondial.

L'interface PMIx, qui propose des diverses solutions portants sur les systèmes de calcul distribué, fournit déjà à sa manière un système de résolution de noms. Cependant, le protocole DNS, qui constitue actuellement la norme du réseau Internet pour la résolution de noms de domaine et d'adresses IP, a fait ses preuves en termes d'efficacité pour l'identification de machines. Ainsi, la connectivité TCP (IPV4,6) est déjà portée par DNS, ce qu'il manque alors est une gestion de clefs étendues pour l'échange d'informations non IP (par exemple pour Infiniband). De plus, le DNS a un scope machine (le plus souvent) et donc il nous faut considérer un stockage temporaire attaché à un job donné, d'où l'idée de *proxy DNS* qui sera développée par la suite. Enfin, particulièrement si le programme est amené à faire aussi du TCP, la PMI peut amener à des résolution DNS par la suite, si on stocke le nom d'hôte dans le clef-valeur (et non l'IP) car faire cette résolution à l'avance est non trivial (déterminer la route vers l'hôte donnée).

Dans ce contexte, l'objectif principal de notre projet consiste à réaliser un proxy DNS dans l'optique de renforcer et d'étendre les capacités de résolution de noms de l'interface PMIx. L'optimisation de notre solution ainsi que la mise en place d'un système de cache est également envisageable. Dans les grandes lignes, nous devons :

- Étudier le protocole DNS et ses implémentations (recherches et documentation) ;
- Créer un client, un serveur et un proxy DNS en utilisant les appels de la *libc* (*getaddrinfo...*) ;
- Ajouter des fonctionnalités supplémentaires à notre solution DNS (utilisation du champ *TXT*) ;
- Effectuer des tests de validation et de performance pour évaluer notre solution.

Les principales motivations pour la réalisation de ce projet sont les suivantes :

- Mettre en évidence le rapprochement entre la résolution de noms et le mécanisme de clé/valeur de la PMIx ;

- Implémenter un proxy DNS fournissant également les capacité de la PMI ;
- Unifier DNS avec la PMI tout en rendant le tout scalable.

2 Implémentations

2.1 Fonctionnement du proxy DNS

Un serveur proxy est un intermédiaire entre le client et l'internet. La requête du client passe par lui et vis versa. Si nous nous plaçons dans le cas où nous avons un client d'un côté, un serveur de l'autre. Ainsi, nous proposons de compléter le DNS en nous insérant au milieu entre le client (le programme MPI) et le serveur (celui de la machine). De cette manière nous créons un espace de nom temporaire (attaché au job) dans lequel nous pouvons fournir les fonctionnalités de PMI et de DNS via un service auto-porté. Pour commencer un ensemble de processus donnés et à terme les processus MPI eux même en peer-to-peer.

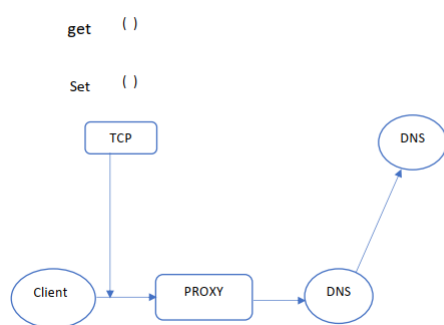


Figure 1 : Figure explicative du rôle d'un bref rôle du proxy

Figure 1 : Figure explicative du

2.1.1 Les structures de données et routines associées

Le proxy DNS est finalement un programme simple, dans un premier temps nous avons adopté un protocole ad-hoc en TCP pour supporter nos requêtes. Le client se connecte au serveur qui maintient une table de hachage des différentes clefs. De plus, si une clef est inconnue, la résolution DNS est tentée (via `getaddrinfo`) afin de retourner une résolution DNS, cette même résolution est stockée dans la table pour optimiser une réponse future, agissant alors comme un cache DNS.

2.1.2 La communication client/serveur

La communication Client Serveur est privilégiée ici dans notre cas pour résoudre le problème de surcharge en fin de rendre plus performant les requête étant donnée que si nous avons ses millier de machines connecter nous

pourrons être confronté à un grand problème de scalabilité. En particulier l'objectif final de ces travaux est de fournir une PMI couplée à un DNS performant. De ce fait, pour répondre à un grand nombre de clients il faut potentiellement un grand nombre de serveurs, encourageant une architecture distribuée.

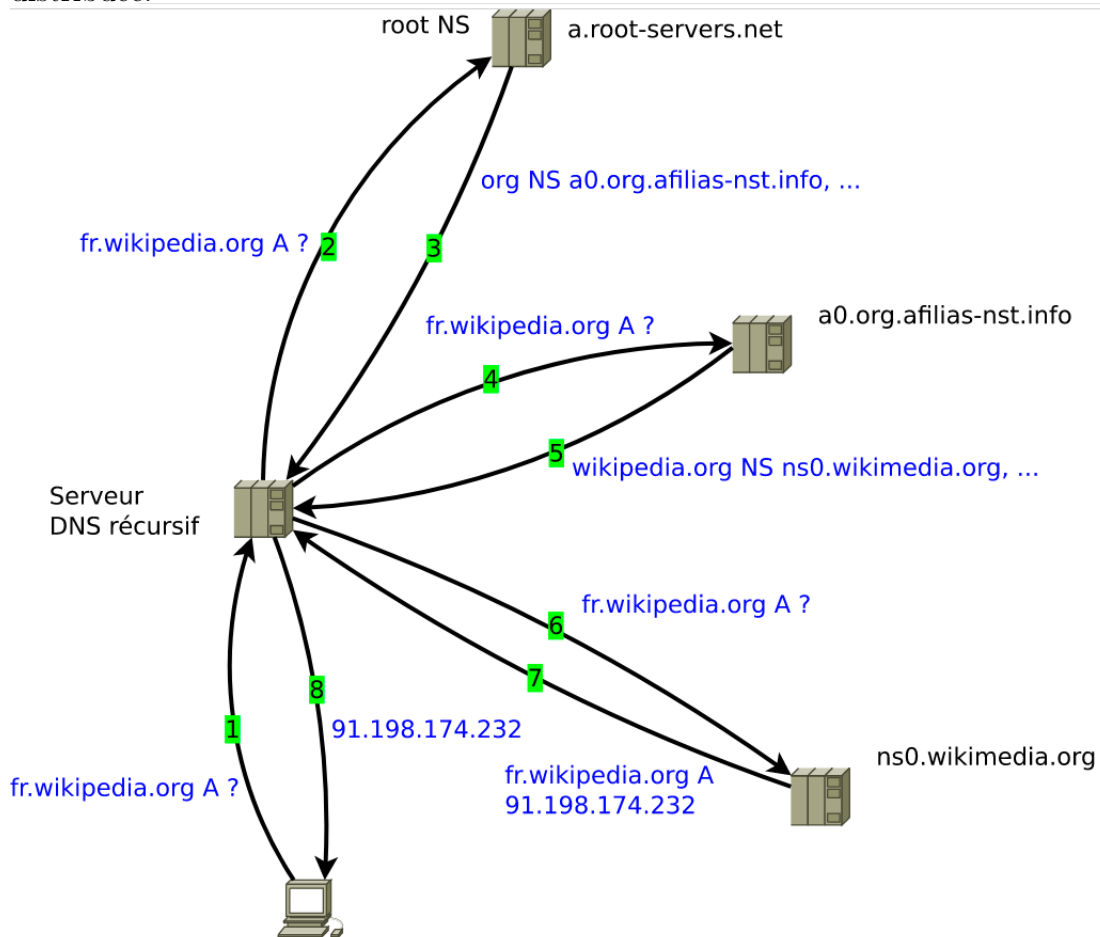


figure 2 : Tirer de wikipedia Cette figure explique comment un client questionne un serveur DNS et de manière récursive comment la réponse est trouver et lui est retransmise à la 8^{me} étape sur la figure1.

2.2 Principe de la table de hachage

La table de hachage est une forme de stockage des données utilisant un algorithme qui est tableau. Comme son nom l'indique, cette table repose sur une fonction de hachage afin de calculer par une opération mathématique une cellule (de manière unique pour une clef donnée) de plus, cette cellule

contient autres tables afin de parer aux collisions(car la table principale est de taille finie). De cette manière la table a une complexité de recherche inversement proportionnelle à sa taille. En effet, la recherche elle en a une complexité au pire des cas une complexité en $O(p)$ avec p le nombre de table inférieur à nombre d'élément/10, ici la table de hachage permet une complexité moyenne en $O(p)$. Ainsi, cette structure de donnée est tout à fait adaptée à l'implémentation d'un stockage clef-valeur.

2.3 Interface Clef-Valeur

C'est une forme d'implémentation qui est basé sur des tableau, facilitant ainsi l'accès à toute la donnée d'un emplacement quelconque de ces donnée à partir d'une information qui peut être clé ou valeur. Elle facilite aussi le stockage des données sous la forme de clé et valeur. Ici elle prend en compte trois principale opérations.

2.3.1 Opération *Get*

Initialement si un programme MPI se connecte, il fait un "get size" et un "get rank". Donc pour un rang, il doit faire un "put" de son identifiant et les autres qui veulent se connecter à lui feront un "get", et c'est PMI qui fait l'échange. Ici, la fonction get est une fonction qui prend une valeur se présentant sous forme de caractère qui pourrait être un clé selon notre implémentation. Le "get" permet donc d'obtenir une valeur qui est retourner par le serveur grâce à un système de fichier ou de cache. La valeur peut être l'IP associée à la clé qui pourrait être à son tour le nom ou un identifiant. Dans le cas ou cette valeur n'existe pas sur le système de fichier ou sur le cache, une requête DNS est lancer pour résoudre le nom.

2.3.2 Opération *Set*

Pour stocker une clé (nom et identifiant) et une valeur nous utilisons la fonction "set". Donc un client en se connectant pourra faire un "set" pour ajouter à la table existante une nouvelle machine (nom et identifiant ou IP).

2.3.3 Opération *Delete*

Enfin, la fonction delete permet de retirer une clef du stockage clef-valeur.

2.4 Système de *log*

Notre système de *log* fonctionne en deux mode : écriture et lecture. Tout d'abord, on déclare deux variables : "char ip" et "char port", ainsi qu'un entier. On demande avec la fonction "fprintf" d'afficher l'adresse IP du client. On utilise la fonction "gets", qui nous permettra d'élire le tableau directement. On fait de même pour le port et le numéro de PC.

2.4.1 Traitement du fichier

Premièrement, nous avons fait une fonction qui écrit dans un fichier fonctionnelle. Ici, le plus important pour la déclaration du fichier, nous faisons appel à "FILE *f" et nous l'affectons à la valeur "NULL". On ouvre notre fichier en mode écriture en faisant appel à la fonction "fopen". Cette dernière va le mode. Voici les différents modes :

- r lecture seule, mais le fichier doit exister
- w écriture seule
- a ajout fin fichier

Il y en a beaucoup comme w+ ou r+ ou a+, mais nous, on utilise le mode r ici pour éviter la fuite de mémoire vis-à-vis de la fonction "gets" qui est un peu dangereuse. Pour créer et écrire dans un fichier, nous avons utilisé le mode "w". Enfin, nous pouvons récupérer les informations existantes dans notre fichier. Pour cela nous utilisons le mode "r" et la fonction "scanf".

3 Choix d'implémentation et Mesure de Performance

3.1 système de cache

3.1.1 table de hachage

Le choix d'une fonction de hachage simple mais efficace réside du faite que nous cherchons à construire un système de clé valeur efficace or nous estimons avoir un très grand nombre de données donc il est important de trouver un algorithme de hachage dont les collisions pourraient considérablement réduite en fonction de l'augmentation de la taille des données. Ce qui répond parfaitement au comportement de notre fonction de hachage qu'on pourra observer sur la figure suivante.

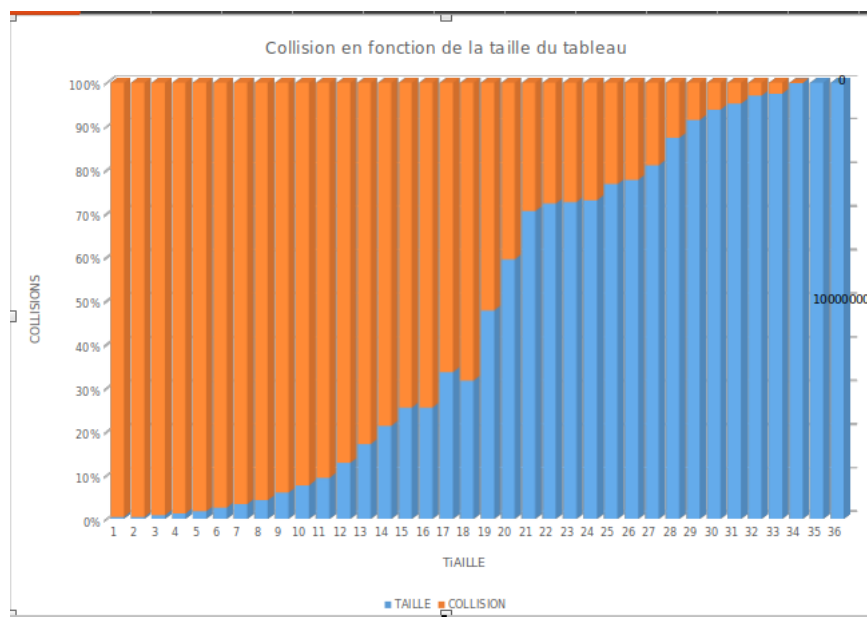


Figure 3 : Explicative de l'évolution des collisions en fonction de la taille du tableau

3.1.2 Sérialisation des données en échange

L'échange de données sans perte de bout en bout étant très important en réseau, nous avons pensés à assurer cette fonctionnalité de transmission sans perte de données.

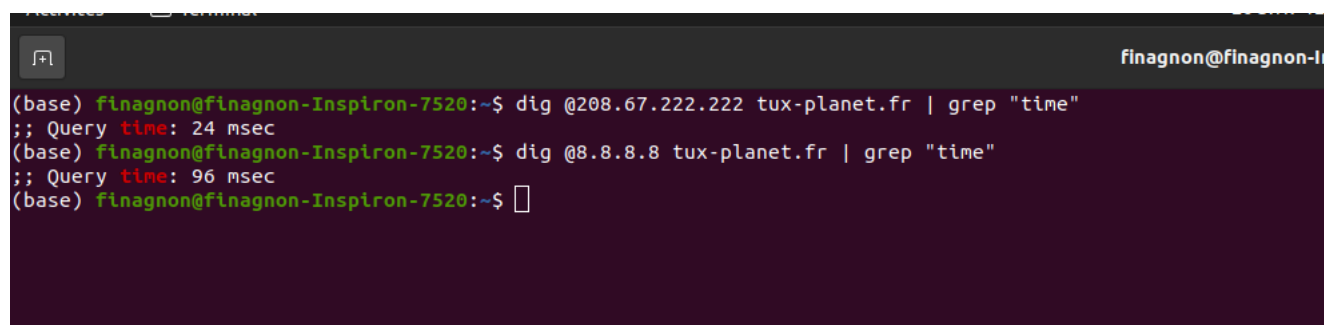
3.1.3 Test de validation

Assurer des entrées et sortir de nos différentes fonctions, il est donc important de les faire validées.

3.2 Parallélisation et Performance

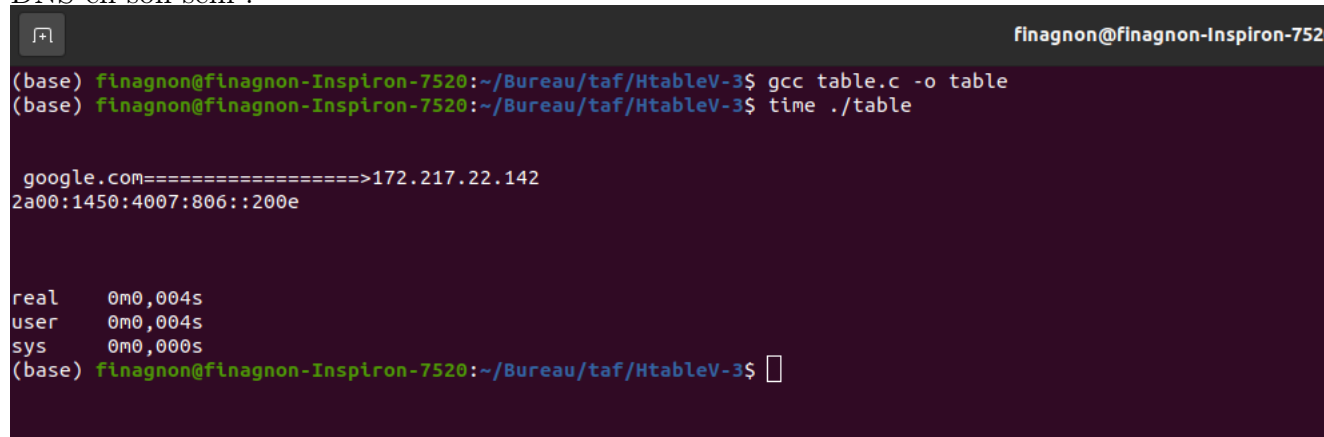
3.2.1 Table de hachage

En séquentiel, Le programme n'assure pas un accès ordonné des différentes ressources, nous avons constaté dans un premier temps qu'ils peut y avoir le stockage de la même donnée à de différents endroit au cour de la même opération d'insertion. Alors donc une suppression non complète des donnée de la table. Pour éviter les effet de bord, des resultats non attendu, il est important de protéger l'accès de données de notre tableau. Cela nous amène à perdre en performance mais que Ce soit la version séquentielle ou parallèle nous avons une meilleur performance qu'une requête DNS :



```
finagnon@finagnon-Inspiron-7520:~$ dig @208.67.222.222 tux-planet.fr | grep "time"
;; Query time: 24 msec
finagnon@finagnon-Inspiron-7520:~$ dig @8.8.8.8 tux-planet.fr | grep "time"
;; Query time: 96 msec
finagnon@finagnon-Inspiron-7520:~$
```

Figure 4 : Montrant le temps d'une requête DNS La figure suivante montrera la version séquentielle de notre programme comportant une requête DNS en son sein :



```
finagnon@finagnon-Inspiron-7520:~/Bureau/taf/HtableV-3$ gcc table.c -o table
finagnon@finagnon-Inspiron-7520:~/Bureau/taf/HtableV-3$ time ./table

google.com=====>172.217.22.142
2a00:1450:4007:806::200e

real    0m0.004s
user    0m0.004s
sys     0m0.000s
finagnon@finagnon-Inspiron-7520:~/Bureau/taf/HtableV-3$
```

Figure 5 : Figure montrant l'exécution séquentielle de notre programme.

Dans cette dernière série de figure nous verrons le temps en réponse de la version parallèle :

```
(base) finagnon@finagnon-Inspiron-7520:~/Bureau/taf/HtableV-3/Version_Parallesiser_htable$ time ./table

google.com=====>142.250.201.174
2a00:1450:4007:806::200e

real    0m0.006s
user    0m0.022s
(base) finagnon@finagnon-Inspiron-7520:~/Bureau/taf/HtableV-3/Version_Parallesiser_htable$
```

Figure 6 : Temps de réponse en parallèle

4 Organisation et déroulement

Notre travail était scindé périodiquement sur le dépôt suivant un fichier pdf vient étayer l'organisation du travail. <https://github.com/sirdelta/ppn-dns>

4.1 Structure du projet

Pour rendre le plus cohérent possible notre solution, nous avons tenu à organiser notre dossier source de sorte à ce que chaque fichier soit spécifique à une activité bien précise dans le déroulement de notre programme. Ainsi, le dossier *src* du projet contient les répertoires suivants :

- Le répertoire *data*, contenant les éléments relatifs à l'envoi et à la réception de paquets de données DNS, ainsi que le stockage de ces données échangées. Il s'agit essentiellement de la définition des structures de données et routines qui permettent de les manipuler.
- Le répertoire *net*, contenant les éléments permettant les échanges de données entre client et serveur DNS. On y trouve les principales fonctions qui orchestrent l'envoi, la réception et le traitement des requêtes aboutissant à la résolution des noms.
- Le répertoire *config*, contenant essentiellement des entêtes dans lesquelles sont définies des variables de tailles et de limites essentielles au bon fonctionnement du programme.

Les principaux fichiers sources du projet sont rangés à la racine du dossier *src*. Le dossier **tests** contient uniquement les fichiers sources des tests de validation sur les méthodes et routines situées dans le dossier *src*. Les exécutables générés sont rangés dans le répertoire *build*.

5 répartition des tâches

Afin de mener à bien notre projet, nous avons tenu à nous répartir les tâches en tenant compte des compétences de chacun en programmation réseau et en gestion de projet. Les tâches ont été attribuées de la manière suivante :

- Y. Kapko : Réalisation de la table de hachage pour le stockage des données échangées ; mise en place d'un système de cache ; test de validation ; étude de performance ; Proposition du rapport.
- S. Koné : Création des structures de données relatives aux requêtes DNS, implémentations d'un client et d'un serveur DNS

- H. MLahara : Réalisation d'un système de *log* pour le suivi du déroulement du programme.

6 Niveau d'achèvement

À l'issue de notre travail, nous sommes parvenu à réaliser un client et un serveur DNS qui communique et s'échanger des données, une table de hachage qui stocke les informations échanger et une première approche pour le système de log. Cependant, notre programme rencontre quelques limites.

6.1 Limites du projet

En effet, la communication client/serveur n'est pas réalisée dans le strict respect de la RFC 1035 décrivant les spécifications du protocole DNS. Il n'y a pas de limite de taille imposée pour les transferts de données d'un nœud du réseau à un autre. De plus, les paquets sont échangés par l'intermédiaire du protocole TCP, et non du protocole UDP, comme précisé dans la RFC. Nous n'avons donc pas pour l'instant proposé un remplacement direct de DNS mais une première itération d'un proxy DNS sur protocole ad-hoc. Cependant, cela permet déjà de vérifier notre capacité à fournir la fonctionnalité clef-valeur, un point que nous allons étendre de manière distribuée dans le futur.

6.2 Travaux Futurs

Pour le prochain semestre, nous envisageons les points suivants :

- L'amélioration du système de cache par la mise en place d'un système plus simple
- Lancement d'application réelle MPI sur notre proxy (implémentation de la PMI1) ;
- L'ajout de tests de validations supplémentaires sur les méthodes et routines ;
- Parallélisation de la table de hachage, La sérialisation de données échanger entre le client et le serveur DNS ;
- Une analyse de performance de notre solution.

Conclusion

Pour conclure, nous pouvons dire dans un premier temps que le protocole DNS est un mécanisme efficace qui permet la résolution de noms et d'adresses IP dans un réseau distribué. Nous nous sommes penchés sur PMIx et DNS afin de chercher les points communs, notre exploration a menée à la réalisation d'un proxy DNS permettant de mimer les fonctionnalités de ces deux composants en même temps. Notre but étant de nous inspirer du meilleur de chacun de ces composants. Notre approche initiale présentée dans ce document a montré la faisabilité de ce point, cependant comme mentionné dans la section précédentes certaines limitations persistent et nous souhaitons les adresser durant le prochain semestre.

Du point de vue des compétences, la réalisation de ce projet nous a permis d'acquérir des connaissances en programmation réseau. En effet, nous avons appris à mettre en pratiques des notions complexes en étudiant des documents techniques et en faisant des recherches sur des domaines très poussés du calcul haute performance.

Références

,https://fr.wikipedia.org/wiki/Domain_Name_System

: *Article sur le DNS.*

16avril202008 : 26.

,<https://datatracker.ietf.org/doc/html/rfc1035>

: *RFC1035 MEMO sur le DNS.*

November 1987 *Network Working Group P. Mockapetris Request for Comments : 1035 ISI*

November 1987

<https://benhoyt.com/writings/hash-table-in-c/>

<https://www.python.org/dev/peps/pep-0456/>

<https://www.youtube.com/watch?v=2Ti5yvumFTU>

<https://www.youtube.com/watch?v=b3dvBA5B4Et> = 2450s

Annexes

Nous avons sur ce dépôt les différents dépôt de notre travail :
<https://github.com/sirdelta/ppn-dns>