Pré Rapport PSAR 2020: [SUJET 2]Observation de la variabilité dans un système réparti : collecte de traces sur la plate-forme PlanetLab

HUYNH Caroline et IKENE Katia
Encadrants: Luciana Arantes et Pierre Sens

March 2020

1 Introduction

Dans un système réparti, la fiabilité et la sûreté sont des contraintes à respecter pour assurer la continuité du service.
Pour cela des algorithmes de traces sont apparus afin de vérifier que le système réponde aux attentes souhaités.
Chaque site représente un noeud dans le réseau. Dans le but de contrôler cette fiabilité, un algorithme de traces est déployé afin d'envoyer des messages de vie(heartbeat) à des instant de temps réguliers. Ces messages sont envoyés à l'ensemble des noeuds du système.
L'objectif de ces traces étant de collecter l'ensemble des informations émisent pas les noeuds.
Pour pouvoir tester ce programme dans un sytème répartis réel, on utilisera le site PlanetLab Europe.
PlanetLab est une plateforme qui est utilisé par de nombreux chercheurs pour tester des algorithmes distribués à grande échelle. Elle regroupe une centaine de sites dans le monde entier.
On va donc utiliser l'outil PlanetLab pour générer nos traces.
À partir de ces traces, des algorithmes de détections de fautes pourront être testés. Cela nous permettra de trouver les sites défaillants au sein de notre système.

2 Approche du Sujet

Dans la suite du projet nous allons utiliser une architecture client/serveur, où chaque site du système représentera à la fois un client et un serveur.Ainsi, chaque site va envoyer à tous les noeuds du réseau , des messages de vie (heartbeat) contenant l'identifiant du site, le numéro du message et l'estampille temporelle de ce dernier(cf. Figure 1).

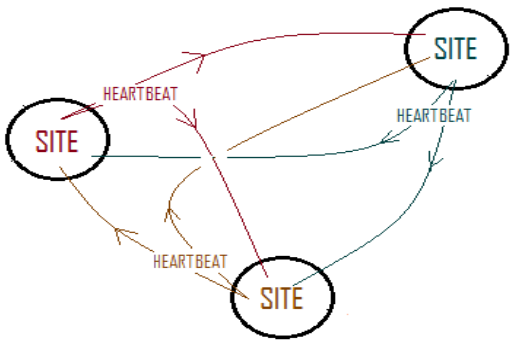

Figure 1: Émission et réception de heartbeat

Ce site aura le comportement d'un client au moment où il emmettra des heartbeats et celui du serveur lorsqu'il sera en attente de la réception des messages des autres noeuds.
Afin de permettre l'envoi et la reception des messages de vie de façons parallèle, il a été décidé de s'inspirer du serveur multi threadé. Chaques site aura un thread client et serveur pour que ces derniers puissent s'éxécuter de manière concurrentes (cf. figure2). Un mecanisme de synchronisation doit alors être mis en place. Les informations de l'ensemble des sites sera gardé dans un fichier config qui sera accessible par tout le monde.
Ce fichier contient l'ensemble des identifants et l'url des sites. Il permet de renseigner au site, les noeuds pour lesquelles il doit envoyer les messages. Les heartbeat seront envoyés par l'intermédiaire du protocole UDP pour faciliter l'envoi en masse de données. Ce protocole de transport à été choisi car on veut éviter les phases de connexions et d'aquittement qui pourrait ralentir la récolte de données. Une socket cliente (respectivement servante) sera implentée dans le client (respectivement serveur).
Dans un premier temps l'algorithme sera testé sur deux machines de la ppti, en se connectant en ssh. Puis sur deux sites de PlanetLab. Une fois qu'on s'est assuré de l'efficacité du programme, il sera déployer en séléctionnant plusieurs sites sur PlanetLab. On pourra être en mesure de collecter des traces contenant l'ensemble des informations des heartbeats reçus, ces informations seront stockées dans un fichierLog propre à chaque site.

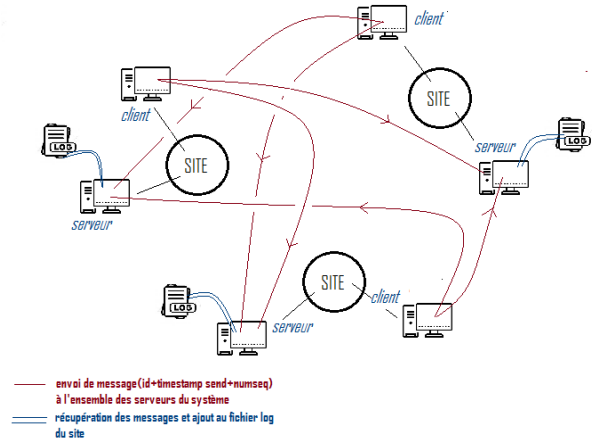

Figure 2: Émission et réception de messsages, architecture client/serveur

L'algorthme de traces sera deployé pour une durée de 24 heures sur la plateforme PlanetLab ce qui nous permettra d'avoir des données suffisantes pour notre analyse , et d'observer au mieux toutes les variabiltés du système.
Après, la collecte des traces vient l'étape de l'observation de la variabilté du système pour ce fait un programme d'analyse sera implanté pour vérifier la latence des messages et détecter les fautes ou eventuellement les craches de machines.

3 Architecture du programme

Pour implémenter l'algorithme de trace, nous avons crée trois classes.
La classe Trace est utilisé comme la classe principale du projet, ce sera à partir de cette classe que nous pourront instancier l'ensemble de nos classes et lancer le programme. Cette classe aura pour variables deux arraylist: une, qui stockera la partie client de chaques sites et l'autre, la partie serveur de chaques sites. L'ensemble des informations des sites du système sera gardé dans une HashMap qui aura comme clé l'identifiant du site et comme valeur, l'url du site.
Dans Trace, nous allons initialiser la map en parcourant le fichier config que l'on aura préalablement crée et enregistré dans notre projet. Ce fichier contiendra l'identifiant du site(unique à chaques sites), son url ainsi que sa localisation (qui nous n'est pas utile pour l'instant). À partir de la map on pourra alors initialiser la liste des clients et des serveurs des noeuds du système. Le ième élément de chacune des listes correspond au ième site dans notre fichier config. Une fois la phase d'initialisation fini, nous pouvons invoquer les méthodes du côté du client et du côté du serveur sur tout nos sites.

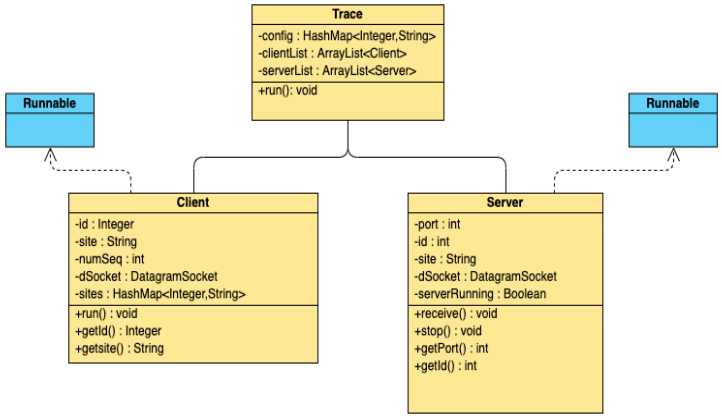

Figure 3: Diagramme UML

Le client et le serveur implémente tout les deux l'interface Runnable.
En effet, comme vu précedemment dans la partie [2. Approche du Sujet], nous avons décidé qu'un site devait être à la fois un client et un serveur. Pour lui permettre d'envoyer des heart-beats et de réceptionner ses messages simultanément.

Dans la classe Client, on stockera l'identifiant du site, le nom du site, le numéro de séquence(qui sera incrémenté à chaque envoie de message) et une map qui gardera l'ensemble des sites pour lesquels nous devons envoyer les messages. Le rôle d'un client étant d'envoyer des messages toutes les 100ms(à redéfinir) pendant N minutes(N est un entier). Le format des messages envoyés est de type: (id + timestamp send + numéro de séquence) ces messages sont envoyés grâce à la classe DatagramSocket, qui utilise le protocole UDP. On parcourt l'ensemble des éléments de la map afin d'envoyer des messages de vie aux serveurs. Lorsque le traitement est finis, le thread s'endort pendant 100ms puis réitère ses instructions jusquà atteindre son temps limite.

La classe Serveur aura pour variable un numéro de port, ici, on a supposé que chaque site aura le même numéro de port. On aura comme pour Client, l'identifiant ainsi que le nom du site. Et il a été ajouté un booléen isRunning, qui est un indicateur pour signaler au serveur que le ramassage de données est terminé et qu'il pourra sortir de sa boucle d'écoute et se terminer. Nous avons crée deux méthodes:
-la méthode stop():qui met le booléen isRunning à false.
-la méthode run(): qui définira le comportement du serveur.
Cette méthode permet au site de récuperer ses messages. Ce traitement est réalisé dans une boucle infini, qui ne s'arrête seulement si on lui envoie un stop. Le serveur récupère le message, puis y ajoute le timestamp received ainsi que le nombre de hop qu'il a fallut pour atteindre la destination. Ces messages seront stockés dans un fichier log. Chaques sites possédera un fichier log qu'il gardera en local, et qui devra être récupéré.

4 Conclusion

Nous avons développé ici l'implémentation de l'algorithme de trace. Ce qui correspond à la première partie de notre projet.
Dans la seconde partie nous devons analyser ces traces. Et proposer une stratégie permettant de distinguer le moment à partir duquel un site est considéré comme en panne. La problématique de cet algorithme étant d'éviter de détecter des pannes ou des pertes là où il n'y en a pas(RTO trop court).