Rapport de stage

Métrologie et supervision de l'infrastructure et des services

Rosendo BONILLA JUAREZ

Observatoire astronomique de Strasbourg | ObAS
IUT nancy Charlemagne Département Informatique
perfSONAR

IUT Nancy Charlemagne
Université de Lorraine
2 ter boulevard Charlemagne
BP 55227
54052 Nancy Cedex
Département informatique

Métrologie et supervision de l'infrastructure et des services

Rapport de stage de LP Administration de systèmes, réseaux et applications à base de logiciels libres
Entreprise : Observatoire astronomique de Strasbourg

Rosendo Bonilla Juarez

Tuteur académique : Lucas Nussbaum
Tuteur professionnel : Thomas Keller
Année universitaire 2017-2018

Remerciements

Tout d'abord, je tiens à remercier l'ensemble de l'équipe de l'Observatoire astronomique de Strasbourg pour m'avoir accueilli et m'avoir permis d'effectuer mon stage pendant ces trois derniers mois.

Je tiens à remercier aussi mes encadrants à l'observatoire Thomas Keller et Christophe Saillard pour le temps qu'ils m'ont consacré tout au long du stage en répondant à toutes mes interrogations et pour leur aide précieuse à la révision de mon rapport de stage et de ma présentation orale.

J'aimerais aussi remercier Lucas Nussbaum, mon référent de stage à l'IUT pour avoir effectué le suivi de mon stage et pour avoir répondu à mes questions relatives à la rédaction des rapports ou à des parties administratives.

Finalement, je remercie aussi Philippe Dosch et Samuel Cruz Lara pour leur aide durant mes cours à l'IUT et pour m'avoir fait confiance et m'avoir encouragé à postuler pour ce stage.

Table des matières

1 Introduction

Dans le cadre de ma formation et pour la validation de ma licence professionnelle, j'ai eu l'opportunité d'effectuer un stage au sein du service informatique de l'observatoire astronomique de Strasbourg, notamment sur la partie administration système et réseau. Pendant trois mois, j'ai travaillé avec mon tuteur M. Keller Thomas et mon co-encadrant M. Saillard Christophe, responsables du service informatique.

Ce stage a particulièrement attiré mon attention parce qu'il est centré sur la supervision du réseau et des services, un sujet passionnant pour moi et qui est lié parfaitement avec ma formation. Ce stage a été mon premier contact avec le monde du travail et ma première expérience professionnelle informatique. Ceci m'a permis de mettre en pratique les connaissances acquises dans mes cours à l'IUT.

Ce rapport a pour but de résumer et retransmettre tout le travail réalisé et tout ce que j'ai apporté à l'observatoire pendant la durée de mon stage.

Dans un premier temps, je présenterai l'observatoire astronomique de Strasbourg, les différentes équipes et services qui le composent ainsi que le département informatique.

Je détaillerai ensuite le déroulement du stage en commençant pour expliquer le contexte et l'environnement du travail. Suivra le planning de mon stage et la description des différentes outils déjà existants et actuellement utilisés.

Puis, je expliquerai de façon chronologique tout le travail réalisé et les étapes suivies ainsi que les résultats obtenus.

J'adresserai finalement une conclusion de tout ce que le stage m'a apporté en ce qui concerne ma vie professionnelle et personnelle.

2 L'observatoire astronomique de Strasbourg

2.1 Présentation générale

L'Observatoire astronomique de Strasbourg est un Observatoire des Sciences de l'Univers, une école interne de l'Université de Strasbourg (Unistra), ainsi qu'une Unité Mixte de Recherche entre l'Université de Strasbourg et le CNRS. [observatoire astronomique, 2018]

FIGURE 1 – Observatoire astronomique de Strasbourg

2.2 Ses équipes de recherche

En tant qu'une partie de l'Université de Strasbourg, l'observatoire gère la parcours de astrophysique du Master en Physique. En tant qu'une unité de recherche, il est structuré en deux équipes scientifiques : GALHECOS et CDS.

2.2.1 GALHECOS

L'équipe « Galaxies, High Energy, Cosmology, Compact Objects & Stars » (GALHECOS) étudie la formation et l'évolution des galaxies et de notre Galaxie dans un contexte cosmologique, au travers de leurs populations stellaires, de la dynamique des étoiles et de la matière noire, et des effets de rétroaction liés notamment à l'activité de leur trou noir central. Elle s'intéresse aux sources galactiques et extragalactiques émettrices en rayons X, objets compacts (étoiles à neutron, naines blanches, etc.) et noyaux actifs de galaxies.

2.2.2 Centre de données astronomiques de Strasbourg (CDS)

Crée en 1972, le CDS héberge entre autres la base de données de référence pour l'identification d'objets astronomiques, ses services — Simbad, VizieR et Aladin — sont largement utilisés par la communauté astronomique internationale. Ses missions sont nombreuses et comprennent entre autres :

- le rassemblement des informations utiles concernant les objets astronomiques
- la mise à jour de ces données
- la distribution de ces données à la communauté internationale
- la mise en place de recherche à propos de ces données

Ses services

Simbad est la base de données de référence mondiale en ce qui concerne la nomenclature et la bibliographie d'objets astronomiques. Possédant l'équivalent de plus de 40 ans de données, Simbad permet aux astronomes de trouver facilement des informations sur plus de 8 millions d'objets grâce à plus de 300 000 références bibliographiques.

VizieR est une base de données regroupant des catalogues d'objets astronomiques. Ces catalogues sont constitués de données relevées durant des missions d'observation et sont ajoutés à VizieR par les documentalistes du CDS. A l'heure actuelle, la base est constituée de plus de 13 000 catalogues.

Aladin est un atlas interactif du ciel permettant de visualiser et de comparer des images du ciel.

2.2.3 Service informatique

Plus précisément, j'ai eu l'opportunité d'effectuer mon stage au sein du service informatique. Il est chargé de maintenir les infrastructures systèmes et réseaux sur lesquelles s'appuient les services des équipes de recherche de l'observatoire, notamment les services du CDS.

Il gère le parc informatique suivant :

- Environ 200 postes de travail principalement sous Linux
- 40 serveurs physiques également sous Linux
- 50 équipements réseau comme des switchs
- Environ un pétaoctet de données stockées

3 Déroulement du stage

Dans cette deuxième section, j'aborde les premiers aspects du stage qui vont permettre d'introduire tout le travail réalisé. Premièrement, le contexte dans lequel le stage se développe et la description de la problématique. Ensuite, la description du sujet de stage et de l'environnement de travail. Après, les différents moyens de documentation que j'ai utilisé pendant le stage ainsi que les personnes avec lesquelles j'ai échangé lors de problèmes ou doutes éventuels. Ensuite, les différentes étapes que j'ai suivies pour atteindre les objectifs du stage. Finalement, une description des outils existants à l'observatoire par rapport au sujet de stage.

3.1 Contexte

L'activité du CDS est fondamentale dans le domaine astronomique. Grâce à ce centre, la communauté internationale des chercheurs et des astronomes peut avoir accès à toutes les informations dont ils ont besoin pour leurs travaux. Ces informations ont été dûment traitées pour les rendre plus faciles à utiliser, ainsi les chercheurs n'ont qu'à s'occuper de leurs investigations en sachant qu'ils ont à sa disposition une infrastructure informatique fonctionnelle et des données prêtes à être utilisées. En effet, pour l'observatoire astronomique de Strasbourg la disponibilité des services qu'il héberge est très importante.

Tout l'ensemble des services du CDS traitent à peu près un million de requêtes par jour provenant de tout le monde. C'est une quantité importante de trafic et il faut donc assurer que les infrastructures systèmes et réseaux soient performantes pour permettre ces échanges de données de façon optimale. C'est pourquoi il est nécessaire de mettre en place des outils de supervision réseau. Cependant, il est bien connu que ce trafic traverse plusieurs domaines et par conséquent la supervision réseau se rend très difficile avec des outils de supervision traditionnels.

3.2 Description du sujet

Le service informatique de l'Observatoire astronomique de Strasbourg conçoit, déploie et maintient les infrastructures systèmes et réseaux sur lesquelles s'appuient les outils développés par les équipes de recherche hébergées, le Centre de données de Strasbourg (CDS) et l'équipe Galaxies et Hautes Énergies.

Le stage proposé est centré sur la problématique de la métrologie et de la supervision des infrastructures et des services dans un environnement multi-domaine. La mission principale du stage est donc consacrée à l'étude et déploiement de perfSONAR, un outil permettant de superviser la performance et connectivité de grosses infrastructures multi-domaine pour garantir une bonne expérience entre les services du CDS et les utilisateurs finaux.

3.3 Environnement de travail

Durant toute la durée du stage, j'ai rejoint le bureau du service informatique situé dans le bâtiment Sud de l'observatoire, même bureau occupé par mon tuteur M. Keller Thomas et mon co-encadrant M. Saillard

Christophe, responsables du service informatique. Ceci a été une point très positif pour moi puisque j'avais la possibilité de leur poser des questions si besoin ou de leur demander des recommandations sur telle ou telle partie du stage. Par ailleurs, ils ont mis à ma disposition une machine dans le bureau avec la liberté de choisir la distribution Linux de ma préférence. J'ai donc décidé d'utiliser Debian Stretch, une distribution que je maîtrisais mieux que d'autres comme CentOS ou Ubuntu car à l'IUT c'était celle que j'utilisais régulièrement. Il faut mentionner que la plupart du personnel, y compris les documentalistes et informaticiens, utilisent Ubuntu comme système d'exploitation principal. Ma machine avait donc Ubuntu installé au départ et j'ai du la formater pour réinstaller le nouveau système.

Avec l'avancement du stage, j'ai eu d'autres machines à ma disposition, notamment des machines virtuelles et une machine physique pour mettre en place perfSONAR. Comme première approche à la supervision avec cet outil, j'ai eu une machine physique pour l'utiliser en tant que nœud de supervision. Cette machine allait être place après dans la salle machine d'un bâtiment de l'observatoire appelé « La grande Coupole ». Deux autres nœuds allaient tourner sur une plate-forme de virtualisation sous VMware, chaque machine virtuelle dans un bâtiment différent pour avoir à la fin trois nœuds de supervision, l'un physique et le reste virtuels. De plus, j'ai eu l'accès à l'interface d'administration de la plate-forme de virtualisation vSphere, dans le cas où j'en avais besoin, par exemple si une des machines s'arrêtait en cas d'une mauvaise configuration ou si je perdais la connexion SSH. Avec cela, je pouvais allumer ou éteindre les machines, supprimer ou en ajouter d'autres ou modifier leurs caractéristiques. Ceci a été très utile surtout au début du stage. Étant donné que je commençais à prendre en main les logiciels, il y avait des fois que je faisais une mauvaise configuration et mon système ne fonctionnait plus.

3.4 Moyens de documentation et contacts

perfSONAR est un logiciel ayant une bonne documentation officielle et en plus une communauté heureuse d'aider tous ceux qui commencent avec perfSONAR. Alors, ma première ressource pour prendre en main ce logiciel a été la documentation officielle. Elle est divisée en plusieurs blocs, commençant par les parties basiques comme son architecture et l'explication de son fonctionnement jusqu'à des parties plus avancées. Il faut mentionner que toute la documentation et les espaces de discussion concernant perfSONAR sont entièrement en anglais, ceci n'a pas été un problème bloquant mais j'avais apprécié de pouvoir trouver des sources en français ou en espagnol bien évidement. Par ailleurs, nous avons eu la chance de pouvoir échanger avec le responsable du déploiement de perfSONAR chez Renater [1]. Il nous a fait une présentation sur perfSONAR au début de la deuxième semaine, ce qui nous a permis de mieux comprendre son fonctionnement et d'arriver à définir une architecture à mettre en place à l'observatoire.

3.5 Planning

Mon stage s'est déroulé en plusieurs étapes :

1. Réseau National de télécommunications pour la Technologie l'Enseignement et la Recherche

1. Prise en main du logiciel : cette étape comporte le renseignement sur les tâches du service informatique de l'observatoire et sur les logiciels à mettre en place.

2. Orchestration d'une infrastructure de test : une étape pour mieux comprendre le fonctionnement des logiciels et faire mes premiers tests à l'aide de conteneurs LXC.

3. Mise en production (infrastructure interne) : adapter mes configurations de l'étape de test à l'infrastructure réelle de l'observatoire.

4. Automatisation de l'administration des sondes : étape de la création de scripts pour faciliter l'ajout et suppression de nœuds et la configuration des tests.

5. Ajout de nouvelles sondes à l'extérieur : étape d'analyse et ajout de nœuds de test en dehors de l'observatoire pour mesurer la performance du réseau depuis l'extérieur.

3.6 Description de l'existant et l'outil choisi

Actuellement, dans l'observatoire il n'existe pas d'outils de supervision réseau. Cela limite la vision de la performance et disponibilité de son réseau. Cette vision est importante pour assurer que le trafic entrant et sortant des services du CDS soit bien transporté tout au long du réseau, des serveurs de l'observatoire jusqu'aux utilisateurs finaux et à l'inverse.

Les seuls outils de supervision qui tournent à l'observatoire sont Centreon et Glu. Centreon est un logiciel de surveillance et supervision réseau basé sur Nagios. Il fournit une interface graphique pour gérer l'administration de Nagios plus facilement. Ensuite, Glu est un outil de supervision applicative crée dans l'observatoire par Pierre Fernique, ingénieur de recherche au CDS. Ce logiciel permet de visualiser d'un coup d'œil l'état fonctionnel de services du CDS. Il fournit des mesures de fiabilité et donne accès à un historique service par service. En cas de panne, il permet de gérer la prise en compte d'une intervention. Cependant, ces outils ne sont pas pour la supervision réseau et ne fournissent pas de données sur la latence ou débit du réseau par exemple.

perfSONAR est donc l'outil choisi, il permet de superviser plus facilement toute une infrastructure réseau multi-domaine. Il est largement utilisé dans le monde entier et compte déjà avec une grande infrastructure globale avec plus de 2000 sondes déployées. Son fonctionnement et sa mise en place sont décris dans la section suivante comme une partie du travail réalisé.

4 Travail réalisé

Cette section a pour but de détailler tout le travail réalisé durant la durée du stage. J'ai suivi le découpage en étapes conformément à la section « planning » pour montrer de façon plus efficace les travaux faits en fonction de l'avancement du stage. Je commence donc pour décrire le logiciel perfSONAR afin de donner une vision globale de ses fonctions. Ensuite, j'aborde mon étape de test qui m'a aidé à mieux

le comprendre. Puis, je décris la mise en place de perfSONAR dans le réseau interne de l'observatoire. Après, je présente une étape d'automatisation aidant à faciliter l'administration du logiciel. Enfin, je décris l'étape d'analyse et ajout de nœuds en dehors de l'observatoire.

4.1 Prise en main du logiciel

Comme première partie du stage, j'ai commencé à me renseigner sur le logiciel perfSONAR à l'aide de sa documentation officielle. J'ai pris certain temps pour la lire et l'analyser. Ensuite, j'ai décidé de tester l'installation de paquets sur ma machine de travail. perfSONAR met à notre disposition les dépôts pour télécharger les paquets sous des versions Debian 8/9 et CentOS 6/7. Il existe aussi la possibilité de télécharger une image CentOS contenant déjà tous les paquets, on n'a qu'à installer le système.

Dans la partie suivante je commence l'explication de l'outil perfSONAR et les différentes éléments qui le composent.

4.1.1 perfSONAR

perfSONAR est un logiciel de supervision pour des réseaux composés par plusieurs domaines. Il permet de détecter des problèmes réseau de bout en bout en mettant à la disposition des administrateurs des informations auxquelles ils n'ont pas facilement accès habituellement.

perfSONAR est très utilisé dans les domaines de recherche. Généralement, ces domaines utilisent de grosses infrastructures réseau qui traversent plusieurs nœuds de communication. L'envoi et réception de données est bien importante entre des chercheurs dans le monde entier, c'est le cas bien sûr de l'astronomie. On dit que la productivité du personnel de recherche est liée, dans une grosse partie, au bon fonctionnement de leurs réseaux informatiques.

Actuellement, il y a plus de 2000 sondes perfSONAR déployées dans le monde entier et il y en a un grand nombre qui permettent de réaliser ses propres tests sans restriction.

Par ailleurs, il existe des outils comme Nagios qui permettent de mesurer des différents paramètres réseau et de détecter des soucis. La différence principale entre ces outils et perfSONAR est que ce dernier a été crée pour superviser des grands réseaux multi-domaine et Nagios, par exemple, sert à superviser un réseau dans un seul domaine. perfSONAR met en place plusieurs sondes de supervision dans des points stratégiques du réseau, c'est-a-dire, dans des parties qui ont un impact direct sur la performance ou qui sont très critiques. Par exemple, derrière un pare-feu, près d'un serveur de calcul, etc. L'objectif est d'arriver à isoler les problèmes le plus possible pour faciliter la mise en œuvre de solutions pertinentes.

En outre, perfSONAR est composé de plusieurs autres logiciels qui aident à stocker les données collectées, planifier le lancement de tests, créer les interfaces d'affichage, etc. On peut diviser ces outils en plusieurs catégories (Figure 2).

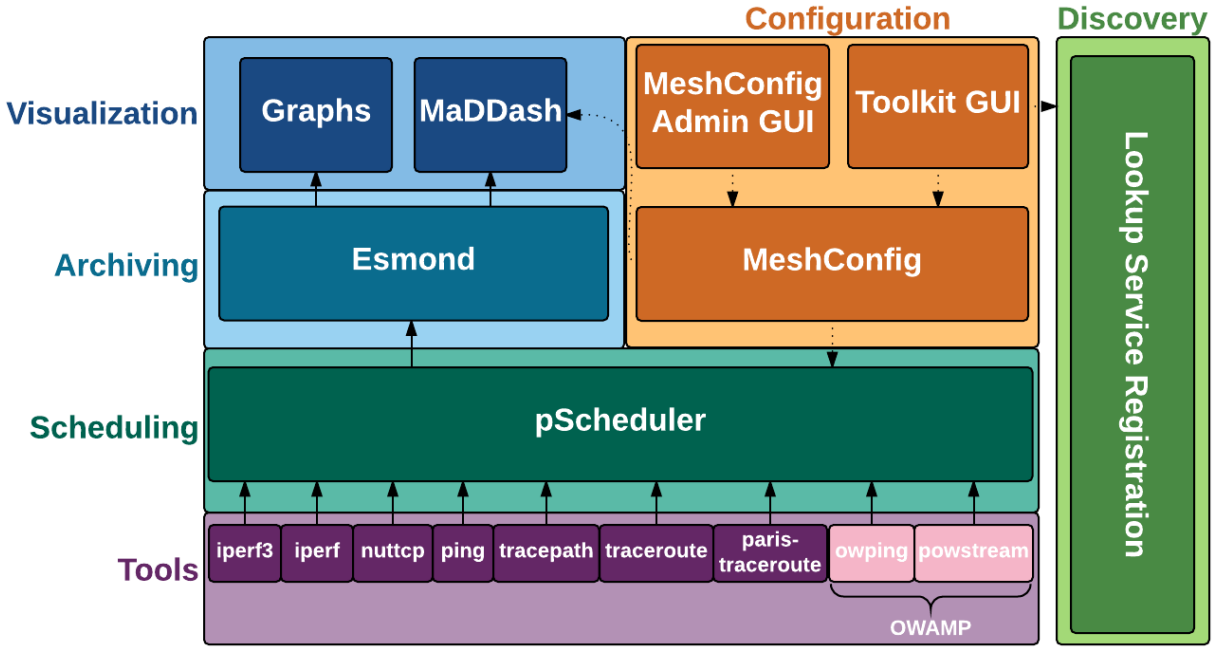


FIGURE 2 – Vue ensemble perfSONAR [perfSONAR, 2018c]

a) Outils de mesure

- OWAMP : est un client en ligne de commande utilisé pour déterminer les paquets perdus et la latence unidirectionnelle entre deux machines. Le grand problème des mesures bidirectionnelles comme ping, est qu'il est difficile d'identifier dans quel sens un problème apparaît. À l'aide d'OWAMP on peut récupérer des données plus illustratives
- iperf3 : est une réécriture de l'outil iperf classique
- iperf2 : est un outil servant à mesurer le débit du réseau
- nuttcp : est un outil aussi pour mesurer la bande passante du réseau
- traceroute : l'outil classique pour identifier les différents chemins traversés pour un paquet
- tracepath : un autre outil pour récupérer les chemins qu'un paquet traverse et en plus mesure le MTU
- paris-traceroute : outil qui essaie d'identifier des chemins en présence de loadbalancers
- ping : l'outil classique pour déterminer la disponibilité, RTT et des paquets perdus
- bwctl : maintenant cet outil est obsolète, il était chargé de la planification du lancement de tests. Cependant, il sert à conserver la compatibilité avec des hôtes tournant des versions obsolètes de perfSONAR.

b) Outils de planification : la couche de planification est composée uniquement d'un outil appelé pScheduler et ce dernier est chargé de :

1. Trouver les temps corrects pour lancer les tests, c'est-à-dire, il identifie le moment précis pour lancer un test en vérifiant que ce dernier n'ait pas d'impact sur d'autres tests puisque ceci pourrait causer des mauvais résultats
2. La collecte de résultats
3. L'envoi des résultats à la couche de stockage

c) Outils de collecte : cette couche est composée d'un seul logiciel appelé esmond chargé de stocker les résultats de tous les tests.

d) Outils de configuration : cette couche définit les tests à lancer ainsi que les instructions pour stocker leurs résultats dans le bon endroit. L'élément principal est le MeshConfig qui est composé par trois parties :

1. MeshConfig Agent : il tourne sur toutes les sondes perfSONAR et il est là pour vérifier que la couche de planification est configurée pour lancer les tests spécifiés. Il cherche des tests locaux définis manuellement et des tests définis à partir d'un fichier de configuration récupéré d'un serveur
2. MeshConfig GUIAgent : il peut lire un fichier de configuration mesh et générer automatiquement un tableau de bord avec l'outil MaDDash et afficher les résultats des tests
3. Meshconfig JSON Builder : c'est un script qui, à partir d'un fichier meshconfig, génère un fichier en format JSON qui sera lu par le MeshConfig Agent et le MeshConfig GUIAgent
4. Toolkit GUI : c'est une interface web d'administration de tests, hosts et d'informations sur le nœud

e) Outils de visualisation : perfSONAR fournit aussi des outils pour pouvoir visualiser graphiquement les données obtenues. Ces sont :

- Graphs : le paquet 'perfsonar graphs' fournit un ensemble de graphiques qui affichent les résultats de test en fonction du temps
- MaDDash : cet outil interroge fréquemment la couche de stockage pour en créer un tableau de bord indiquant la performance entre les éléments de chaque tableau (chaque tableau représente un test définit)

f) Outils de découverte : il existe un outil appelé Lookup Service (LS) Registration Daemon qui permet d'enregistrer l'existence d'un nœud dans une zone publique ou privée et de pouvoir en repérer d'autres.

4.1.2 Les tests

Les tests sont une partie fondamentale de perfSONAR, ils permettent de récupérer les résultats des mesures et pouvoir identifier des problèmes de performance entre les parties concernées (entre deux nœuds de supervision par exemple). Il existe trois catégories :

1. Exclusive : ce sont de tests qui ne peuvent pas être lancés au même temps qu'un autre test exclusif ou normal. Par exemple, les tests de débit.

2. Normal : ces tests peuvent s'exécuter au même temps que d'autres tests normaux ou que ceux du type arrière-plan mais pas au même temps que les tests exclusifs.

3. Arrière-plan : ces tests peuvent bien vivre avec n'importe quel autre type de test. Par exemple, les tests de latence, ping et traceroute.

Il faut mentionner que tous ces types de tests nécessitent du protocole NTP [2] pour amener des résultats cohérents. Ce protocole permet une synchronisation de l'horloge entre les machines concernées dans le lancement d'un test. Plus précisément, je décris dans la partie suivante comment ce protocole impacte les résultats des tests.

4.1.3 NTP

Network Time Protocol (« protocole d'heure réseau ») ou NTP est un protocole qui permet de synchroniser, via un réseau informatique, l'horloge locale d'ordinateurs sur une référence d'heure. Autrement dit, il permet de synchroniser l'horloge d'un ordinateur avec celle d'un serveur de référence. NTP est un protocole basé sur UDP et utilise le port 123. [Wikipedia, 2018b]

Ce protocole est très important pour le bon fonctionnement de perfSONAR, surtout pour les outils de test comme OWAMP et BWCTL. Ces derniers envoient des paquets contenant l'heure actuelle pour pouvoir établir la communication. Il est important que les deux nœuds soient synchronisés pour obtenir des résultats les plus précis possibles. En effet, des temps désynchronisés produisent de mauvais résultats. Par exemple, si l'heure d'un nœud est désynchronisé, perfSONAR et MaDDash peuvent montrer des erreurs de performance entre les nœuds concernés même s'il n'y en a pas du fait d'un grand écart entre les heures de chaque nœud. Cependant, ces outils permettent de désactiver la vérification de l'heure et par conséquent NTP n'est plus nécessaire mais ceci aura un vrai impact sur les résultats, ce n'est donc pas conseillé.

En outre, perfSONAR conseille de configurer le démon de NTP pour se synchroniser avec les quatre ou cinq serveurs NTP le plus connus, qui par défaut sont ceux auxquels la machine hôte est rattachée. Cette configuration garantit une meilleure disponibilité dans le cas où un serveur ne se trouve pas disponible. D'autres recommandations sont : ne pas choisir des serveurs NTP très éloignés car cela pourrait impacter

2. Network Time Protocol

sur la latence en l'incrémentant, les serveurs doivent se trouver dans des réseaux différents pour assurer la disponibilité, etc.

4.1.4 Toolkit GUI

perfSONAR propose un moyen pour administrer les informations des tests et du protocole NTP : un interface web connu comme Toolkit GUI.

Il y a plusieurs types d'installation disponibles par rapport à des besoins particuliers (Figure 3, page 15). Celle choisie pour l'observatoire est l'installation perfsonar-toolkit. Cette installation inclue tous les outils de mesure, de planification et de stockage et en plus l'interface web Toolkit GUI.

Une fois l'installation faite, une interface web (Figure 4, page 15) affiche les services perfSONAR tournant sur la machine, les tests définis s'il y en a et les informations sur le système ou NTP. Cette interface permet de définir les tests qui se lanceront périodiquement entre les nœuds choisis et les résultats sont affichés à l'aide de graphiques.

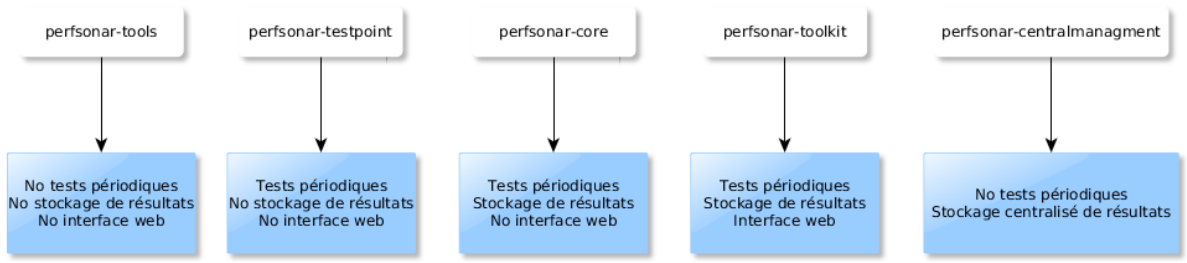


FIGURE 3 – Types d'installation

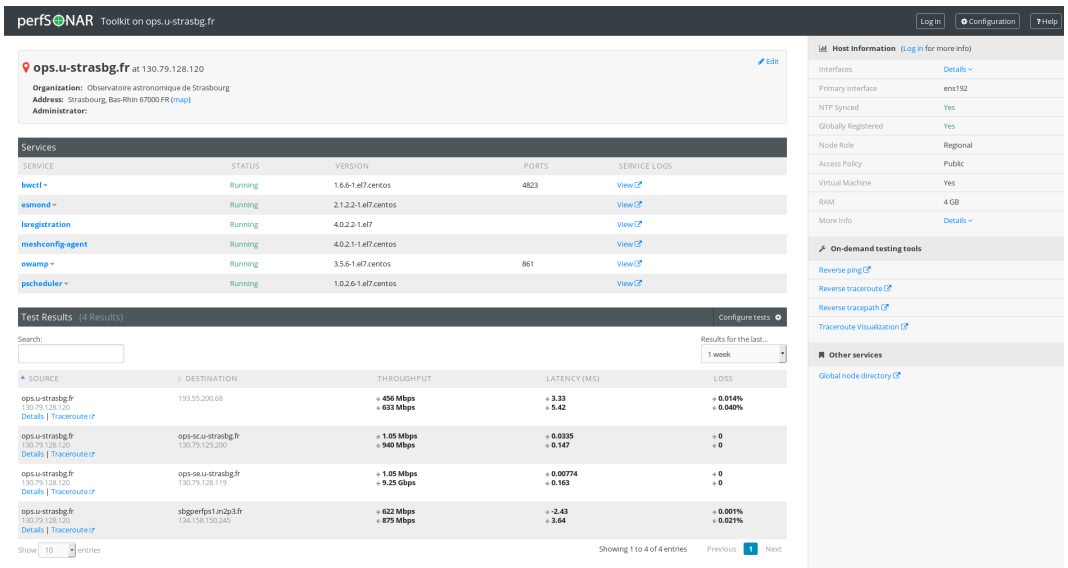


FIGURE 4 – Interface web Toolkit

4.1.5 Services

L'interface Toolkit rassemble les services actifs de perfSONAR en affichant leur état, version, ports qui utilisent et leurs fichiers de logs.

Les plus importants sont : BWCTL, esmond, lsregistration, meshconfig-agent, OWAMP et pscheduler.

bwctl

BWCTL maintient la compatibilité avec des sondes perfSONAR tournant une version plus ancienne. Il était chargé de planifier le lancement des tests. Désormais, c'est pScheduler qui le fait. Cependant, BWCTL peut servir encore pour lancer des tests à la main.

esmond

esmond permet de collecter, stocker, visualiser et analyser une très grande quantité de données de séries temporelles. [esnet, 2018a]

Il utilise une modèle de stockage mixte, il stocke les séries temporelles à l'aide de Cassandra [3] et les métadonnées à l'aide de PostgreSQL ou SQLite.

lsregistration

Les données d'emplacement servent à repérer la sonde publiquement. Le service perfsonar-lsregistrationdaemon permet d'enregistrer ou mettre à jour ces données pour que d'autres sondes puissent s'en servir et l'interroger.

Le Lookup Service Directory [4] permet de placer dans une carte mondiale toutes les sondes publiques enregistrées.

Voici quelques images du Lookup Service Directory :

(a) Carte perfSONAR (b) Emplacement du nœud observatoire

FIGURE 5 – Lookup Service Directory

3. Base de données du type NoSQL - http ://cassandra.apache.org/
4. Interface web affichant tous les nœuds perfsonar http ://stats.es.net/ServicesDirectory/

pScheduler

pScheduler est le substitut de BWCTL à partir de la version 4.0 de perfSONAR. Il est responsable de planifier et gérer le lancement de tests à l'aide de tâches. Un tâche consiste à une unité de travail comportant un test à lancer et les informations de planification.

4.1.6 Sécurité

Dans cette partie je voudrais aborder la sécurité réseau parce que le déploiement de perfSONAR représente une ouverture de plusieurs ports et l'inclusion de nouveaux éléments dans le réseau, c'est le cas des nœuds de supervision. Chaque outil a besoin d'un port ou même de certaines plages de ports (Figure 6).

perfSONAR Tools Ports		
Tool	**TCP ports**	**UDP Ports**
owamp	861	8760-9960
pscheduler	443	
iperf3	5201	
iperf2	5001	
nuttcp	5000, 5101	
traceroute		33434-33634
simplestream	5890-5900	
ntp		123
bwctl	4823, 5001-5900, 6001-6200	5001-5900, 6001-6200

(a) Ports pour les outils de mesure

perfSONAR Toolkit Ports	
Tool	**TCP ports**
management interface	80, 443
esmond	80, 443
Lookup Service	8090

(b) Ports pour le Toolkit GUI

FIGURE 6 – Ports nécessaires pour perfSONAR [perfSONAR, 2018a]

Le perfSONAR Toolkit définit par défaut un ensemble de règles iptables pour n'autoriser que ces ports. En plus, il inclue fail2ban [5] pour détecter certaines attaques.

Ensuite, perfSONAR offre la possibilité de traiter des certificats lorsque les nœuds récupèrent le fichier de configuration centralisée.

En outre, les outils comme OWAMP, BWCTL et pScheduler ont un fichier 'limits.conf' pour définir des règles lors de l'exécution d'un test. Avec cela, ils peuvent refuser le lancement de tests pour certaines plages d'adresses, définir la taille du trafic permis et plusieurs autres paramètres. Enfin, perfSONAR propose de télécharger un fichier contenant la liste de tous les réseaux institutionnels et de recherche et configurer pScheduler et BWCTL pour l'utiliser et de cette manière réduire les problèmes de sécurité.

4.1.7 MeshConfig File

Comme dernière partie de cette étape de prise en main, j'aborde de façon générale le fichier principal de configuration : le fichier meshconfig.

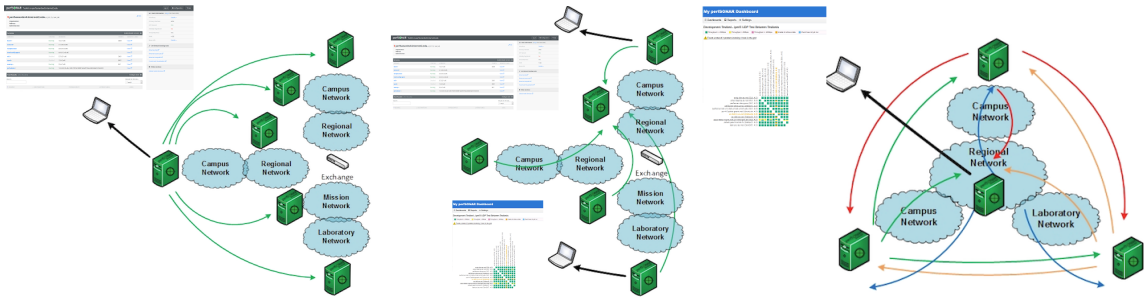
Tout d'abord, il faut mentionner que perfSONAR propose trois architectures de déploiement : island, beacon et mesh (Figure 7 - Page 18). La première consiste à avoir un seul nœud qui va en interroger d'autres. La deuxième rend disponible un nœud pour que d'autres nœuds puissent s'en servir et faire des

5. Outil pour loguer les activités étranges sur un système, par exemple plusieurs essais échoués de se loguer en SSH

tests. La dernière permet d'avoir un ensemble de nœuds travaillant de manière coordonnée, un serveur publie un fichier de configuration contenant les groupes et les tests à effectuer et les clients le récupèrent pour se joindre au maillage.

Pour les besoins de l'observatoire on a décidé de mettre en place un maillage pour avoir des tests entre les bâtiments et avec des nœuds en dehors de l'observatoire. Pour l'architecture 'mesh' ou 'maille', il nous faut donc le fichier de configuration centralisée.

Ce fichier est très important pour la mise en place de perfSONAR à l'observatoire car toute la configuration est gérée par ce fichier.



(a) Island (b) Beacon (c) Mesh

FIGURE 7 – Types de déploiement Source [perfSONAR, 2018b]

4.2 Orchestration d'une infrastructure de test

Cette section a pour but d'expliquer les étapes suivies pour mettre en pratique les connaissances acquises durant la phase de renseignement. Ici, je décris plus en détail les parties du fichier meshconfig, mes premiers tests et les problèmes rencontrés.

4.2.1 LXC/LXD

Avant toute mise en production, il est conseillé de faire un prototypage pour éviter d'endommager l'environnement de production courant. Je suis donc parti sur l'utilisation de conteneurs LXC pour faire mes tests.

Définition

LXC, contraction de l'anglais Linux Containers est un système de virtualisation, utilisant l'isolation comme méthode de cloisonnement au niveau du système d'exploitation. Il est utilisé pour faire fonctionner des environnements Linux isolés les uns des autres dans des conteneurs partageant le même noyau et une plus ou moins grande partie du système hôte. Le conteneur apporte une virtualisation de l'environnement d'exécution (processeur, mémoire vive, réseau, système de fichier...) et non pas de la machine. Pour cette raison, on parle de « conteneur » et non de « machine virtuelle ». [Wikipedia, 2018a]

LXD est un outil d'administration qui complète LXC. Avec une seule commande intuitive il permet de contrôler tous les conteneurs. Ces conteneurs sont gérés par le réseau de façon complètement transparente grâce à une interface REST. Il marche aussi à grande échelle, s'intégrant avec OpenStack. [LinuxContainers, 2018]

L'environnement de test

La création de conteneurs LXC est très facile. J'en ai créé deux, l'un étant le serveur perfSONAR et l'autre le client. Que ce soit le serveur ou le client, tous les deux contiennent le Toolkit GUI et peuvent exécuter des tests.

LXD assigne des adresses privées 10.x.x.x. J'avais donc le serveur avec une adresse 10.254.11.128 et le client avec une adresse 10.254.11.69. Puis, j'ai installé CentOS 7 et le perfsonar-toolkit sur chacune des machines. Enfin, l'accès à l'interface d'administration de perfSONAR est possible.

4.2.2 Création du premier fichier MeshConfig

Pour l'infrastructure de test, il a fallu créer un premier fichier meshconfig. Cette étape a été très utile pour me familiariser avec les directives, tester mes configurations et résoudre les erreurs rencontrées.

Tout d'abord, il est important de mettre une bonne description pour le fichier de configuration. La directive 'administrateur' peut être ignorée, elle n'est pas forcement requise.

```
1  description PerfSONAR Observatoire Mesh Config
2
3  <administrator>
4    name        Rosendo BONILLA
5    email       bonilla.juarez.rosendo@gmail.com
6  </administrator>
```

Ensuite, une organisation est composée de nœuds (directive site) et chaque nœud d'informations comme son emplacement et son adresse IP.

```
7  <organization>
8      description     Observatoire Astronomique de Strasbourg
9
10     <site>
11         <host>
12               description Node 1
13               address node1.lxd
14         </host>
15     </site>
16 </organization>
```

Puis, les paramètres de chaque test son définis à l'aide de la directive test_spec. Chaque test a des paramètres différents. Pour ce premier fichier il n'y a que deux tests, un test ping et un test de latence mais bien évidemment on peut y en rajouter d'autres.

```
<test_spec owamp_test>
  type              perfsonarbuoy/owamp
  packet_interval   0.1
  loss_threshold    10
  session_count     10800
  sample_count      600
  packet_padding    0
  bucket_width      0.0001
  force_bidirectional 1
</test_spec>

<test_spec ping_test>
  type              pinger
  test_interval     300
  packet_count      10
  packet_interval   1
  packet_size       1000
</test_spec>
```

Après, il faut définir les groupes rassemblant les nœuds par rapport à nos besoins. Il existe quatre types de groupe : mesh, disjoint, ordered mesh et star. Mesh permet à tous les nœuds de faire de tests vers tous les nœuds. Disjoint, un ensemble de nœuds vers un autre ensemble de nœuds. Ordered mesh, similaire à mesh mais dans ce cas l'ordre défini est important. Star, un seul nœud central vers plusieurs autres nœuds. Ce dernier peut bien être substitué par un groupe 'disjoint' avec un seul 'membre A' et plusieurs 'membre B'.

```
<group mesh_test>
    type        mesh

    member      node1.lxd
    member      node2.lxd
</group>
```

La dernière partie relie les groupes et les tests définis précédemment.

```
<test>
  description       OWAMP Testing
  group             mesh_test
  test_spec         owamp_test
</test>

<test>
  description       Ping Testing
  group             mesh_test
  test_spec         ping_test
</test>
```

4.2.3 Premiers tests

Une fois que le fichier de configuration est créé, il faut le convertir en format JSON et ensuite le publier sur un serveur pour le rendre accessible et que d'autres nœuds puissent le récupérer.

Finalement, il suffit d'aller sur chaque nœud et modifier son fichier meshconfig-agent.conf placé dans /etc/perfsonar/ en y rajoutant le chemin vers le fichier JSON.

Un redémarrage du service perfsonar-meshconfig-agent est requis pour forcer la prise en compte des modifications. Avec tout ceci, les fichiers meshconfig-agent-tasks.conf de chaque nœud vont être configurés automatiquement avec les tests définis dans le fichier meshconfig.json.

Maintenant, tous les nœuds sont configurés pour lancer des tests vers les autres à un moment donné.

4.2.4 Problèmes rencontrés et solutions

Syntaxe

Au début les erreurs fréquentes étaient liées à la syntaxe puisque je ne connaissais pas de toute la structure et les instructions du fichier de configuration. Cela m'a pris beaucoup de temps. À chaque fois, je devais faire face à plusieurs problèmes de ce genre et ensuite prendre de temps pour aller chercher dans les fichiers de logs, me renseigner sur internet pour enfin les corriger.

NTP

Avant la création de conteneurs, j'avais lu sur Internet que l'on devait les créer en mode privilégié pour éviter des problèmes et j'ai donc fait comme cela. Cependant, NTP n'arrivait pas à se synchroniser et les logs ne montraient aucune erreur. Par ailleurs, la date était la même que dans la machine hôte. Enfin, j'ai décidé de changer en mode non privilégié et le problème est disparu.

Fichier limits.conf

Au début, aucun test ne se faisaient pas même si je ne rencontrais aucun problème dans les logs. Les configurations étaient bonnes mais j'avais toujours le problème.

Après quelques recherches sur le forum de perfSONAR et dans sa documentation, j'ai trouvé quelque chose qui m'a aidé même si ceci ne correspondait pas directement à mon problème.

Il existe un fichier limits.conf qui appartient à l'outil pscheduler. Ce fichier est chargé d'autoriser ou refuser le lancement de tests pour certains groupes d'hôtes, de spécifier les tests permis ou leurs paramètres. Dans ce fichier j'ai trouvé que toutes les plages d'adresses privées étaient bloquées. Évidemment, mes conteneurs LXC appartenaient à un réseau privé, en conséquence les tests échouaient. En plus des plages d'adresses, là-dedans, perfSONAR bloque certaines groupes d'hôtes considérées comme pirates informatiques.

Pour mes tests, j'ai enlevé la plage d'adresses 10.x.x.x du fichier limits.conf et les tests ont été lancés.

4.3 Mise en production (infrastructure interne)

Une fois j'avais un fichier de configuration centralisée fonctionnel, j'ai décidé de l'adapter pour l'architecture interne de l'observatoire et commencer à avoir des données sur la performance du réseau.

Cette section aborde donc la description de cette architecture, la construction du fichier meshconfig correspondant, l'installation d'un outil pour mieux visualiser les résultats des tests et les problèmes rencontrés.

4.3.1 VMWare vSphere

VMware vSphere est un logiciel d'infrastructure de Cloud computing de l'éditeur VMware, c'est un hyperviseur de type 1 (Bare Metal), basé sur l'architecture VMware ESXi. [Wikipedia, 2018c]

La gestion de ce serveur hôte peut se faire via plusieurs possibilités : par le navigateur Web avec une connexion directe, par une console cliente avec une connexion directe ou par un outil de gestion centralisée nommé VMware vCenter Server qui permet d'administrer l'ensemble des machines virtuelles, des hôtes physiques, de leurs ressources et des options de l'environnement (High Availability, vMotion, Storage vMotion, Distributed Resource Scheduler, Fault Tolerance...) depuis une seule console. [Wikipedia, 2018c]

L'observatoire possède une plate-forme de virtualisation administrée à l'aide de VMware vSphere. C'est un logiciel que je ne maîtrisais pas du tout en début de stage et même pas assez bien à présent. Cependant, j'ai les connaissances nécessaires pour travailler avec. En effet, je sais juste créer ou supprimer des machines virtuelles, gérer leur paramètres de façon générale, etc. Par ailleurs, pour le stage je l'ai utilisé surtout au début pour l'étape de tests. Puis, tous les travaux je les faisais à l'aide de SSH.

4.3.2 Description de l'infrastructure perfSONAR

Même si perfSONAR est fait pour la supervision de réseaux multi-domaine, le service informatique a décidé de le mettre en place aussi en local pour mesurer la latence et bande passante entre ses bâtiments. Pour le moment le réseau est tout simple avec un seul domaine mais cela changera dans les prochains mois et ce réseau passera a être découpé et routé.

Pour l'infrastructure perfSONAR interne pensée pour l'observatoire il y a un nœud placé dans chacun des trois bâtiments. La distribution est la suivante :

1. Nœud Sud (central) : ce nœud héberge toute la configuration et le serveur MaDDash [6]. Il est virtualisé sur un serveur dans le même bâtiment que le bureau informatique. Le nom DNS défini pour ce noeud est ops.u-strasbg.fr

2. Nœud Est : un nœud de supervision virtualisé sur un serveur dans le bâtiment Est. Le nom DNS défini pour ce noeud est ops-se.u-strasbg.fr

3. Nœud Coupole : un nœud de supervision sur une machine physique dans le bâtiment de la Grande Coupole. Le nom DNS défini pour ce noeud est ops-sc.u-strasbg.fr

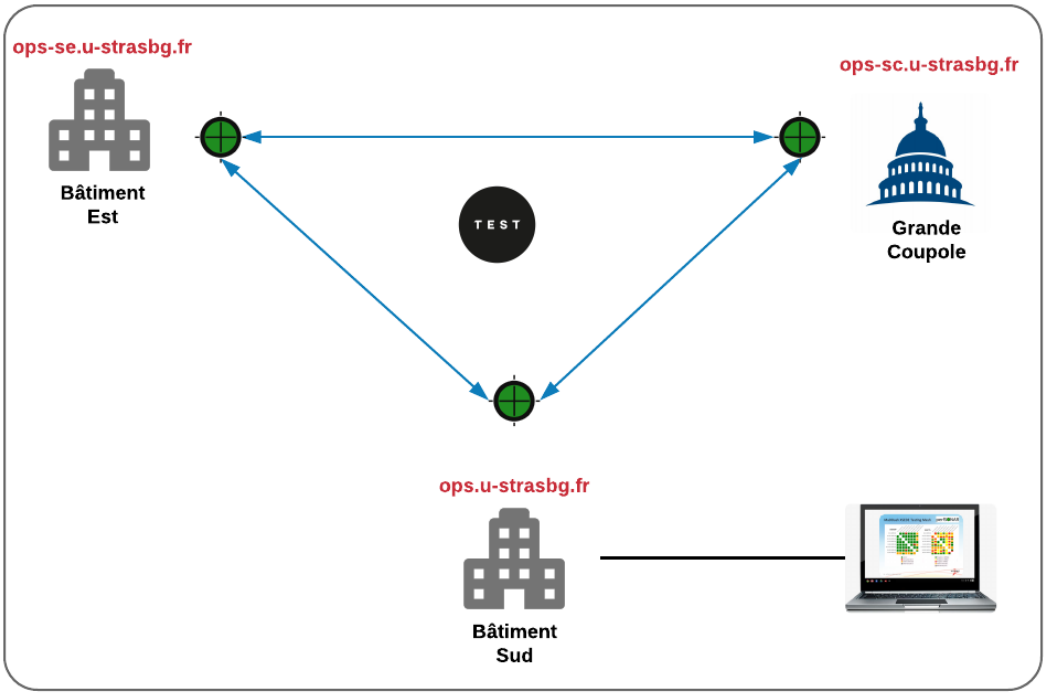


FIGURE 8 – Architecture perfSONAR interne

6. Logiciel pour l'affichage et analyse des résultats entre plusieurs nœuds perfSONAR

4.3.3 Création du fichier MeshConfig

La mise en place de perfSONAR sur des conteneurs LXC m'a aidé à bien comprendre certains concepts. La création du nouveau fichier meshconfig a été plus facile.

D'après l'infrastructure décrite au-dessus, il faut créer un groupe pour les nœuds en interne. Il est de type mesh parce que l'on veut que tous fassent des tests vers tous.

```
<group obas_interne_mesh>
    type        mesh

    member      ops.u-strasbg.fr
    member      ops-se.u-strasbg.fr
    member      ops-sc.u-strasbg.fr
</group>
```

Concernant les nœuds, il n'y a pas un grand changement par rapport au fichier de test. Il a fallu juste changer les noms et les adresses du fichier de test et ajouter un bloc pour le nouveau nœud.

Par rapport aux tests, j'en ai rajouté d'autres comme ceux de débit TCP et UDP. Avec cette configuration, on peut déjà visualiser les résultats de chaque nœud à l'aide des graphiques automatiquement crées par perfSONAR. Cependant, on peut aller un peu plus loin avec la mise en place de MaDDash [7] qui permet de visualiser tous les tests et nœuds dans un même endroit.

4.3.4 MaDDash

MaDDash est un logiciel pour l'affichage et analyse de résultats actuellement développé comme une partie de perfSONAR [esnet, 2018b]. La particularité de ce logiciel est qu'il permet l'affichage de résultats en deux dimensions dans un tableau de bord. Ces grilles permettent d'identifier plus efficacement des problèmes entre certains nœuds.

Son installation est facile. On n'a qu'à ajouter un dépôt et lancer install maddash-server avec yum ou apt selon le système. Sa configuration est gérée par le fichier maddash.yaml placé dans /etc/maddash/maddash-server/. Cette configuration peut être faite à la main mais cela est censé être un peu plus difficile si on ne maîtrise pas bien le logiciel. Heureusement, perfSONAR fournit un service qui crée automatiquement sa configuration à partir d'un fichier meshconfig.json, le service perfsonar-meshconfig-guiagent. À chaque modification du fichier meshconfig-agent.conf, ce service est redémarré.

Enfin, on obtient un beau tableau de bord comme celui de ci-dessous dans la Figure 9 :

7. Monitoring and Debugging Dashboard

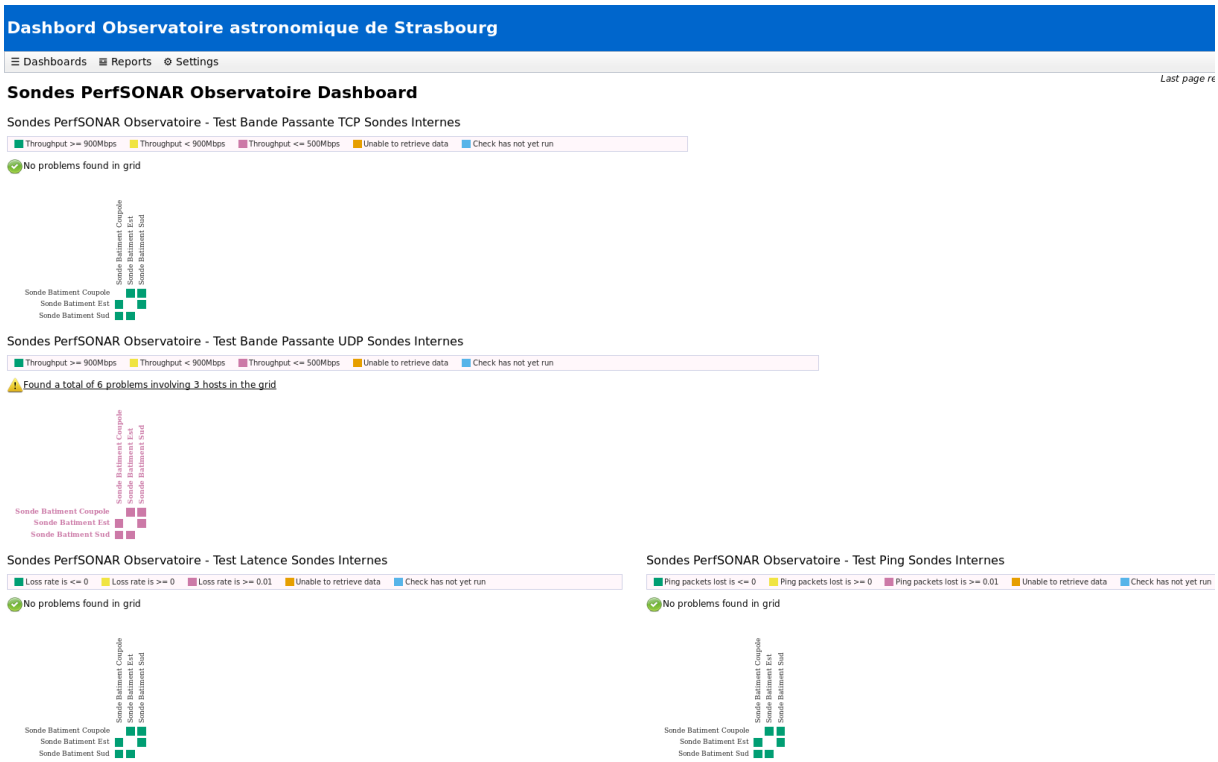


FIGURE 9 – Tableau de bord Observatoire

À partir de ce moment, le tableau est disponible sur l'URL http ://ops.u-strasbg.fr/maddash-webui.

Ce tableau affiche l'état des tests entre les nœuds. Chaque carreau représente la relation entre deux nœuds. Cela veut dire qu'il y a des tests dans un sens et dans l'autre. On voit les trois bâtiments configurés pour lancer des tests entre eux.

En cliquant sur un carreau, le graphique correspondant est affiché. En plus, s'il y a des problèmes, il affiche aussi les possibles solutions et d'autres informations, ce qui est connu dans perfSONAR comme des rapports.

Un exemple d'un graphique s'affiche ci-dessous dans la Figure 10 :

FIGURE 10 – Graphique test entre ops-se.u-strasbg.fr et Renater

Ce graphique affiche les résultats des tests effectués à ce moment-là. Par exemple, les lignes en bleu, elles correspondent aux résultats des tests de débit TCP. Elles affichent que l'observatoire a une bande passante à peu près symétrique entre 200 Mo et 800 Mo pour la tranche de temps sélectionnée.

4.3.5 Problèmes rencontrés et solutions

Interface web Toolkit Web

Lorsque j'avais installé les paquets perfSONAR sur les machines virtuelles et une machine physique, je me suis rendu compte que l'interface du Toolkit restait bloquée dans le chargement de services et des tests. Dans les fichiers de logs d'Apache, il affichait une erreur liée au niveau d'exécution. Après quelques essais de changer le niveau d'exécution on a décidé de prendre l'image ISO fournit par perfSONAR pour voir si le problème venait d'une manque de paquets. Dès que l'on a vu que l'image a fonctionné, on l'a installée dans toutes les machines.

MaDDash

J'ai commencé à travailler avec MaDDash dans cette étape, dans mes conteneurs j'avais juste réussi à créer un fichier meshconfig qui fonctionnait.

Le premier problème rencontré était lié d'ailleurs au fichier meshconfig. Un message d'erreur disant que la création de la configuration avait échouée parce que le service n'avait pas trouvé les blocs contenant les chemins vers les résultats des tests. Étant donnée que j'avais pris une configuration très minimale pour mes tests dans les conteneurs, elle ne contenait pas ces blocs (exemple de ces blocs, voir Annexe 6.1, page 40, lignes 12 à 34).

Ensuite, le fichier maddash.yaml avait comme propriétaire à 'root' et MaDDash ne réussissait pas à y écrire. J'ai du changer le propriétaire à 'maddash'.

Puis, j'ai rencontré un problème qui m'a pris plusieurs jours à résoudre parce que je ne trouvais pas de réponse sur internet et les logs sur l'erreur n'étaient pas très descriptifs. Lorsque je redémarrais le service concernant MaDDash je rencontrais l'erreur suivante : Problem reading remote meshes. En fait, MaDDash a un script en Perl qui génère toute la configuration pour le tableau de bord de MaDDash. Il cherche la définition de tests sur chacun des nœuds concernés dans la maille. Il vérifie donc que le bloc contenant le chemin du fichier JSON est défini dans le fichier /etc/perfsonar/meshconfig-agent.conf sur chacun d'eux, y compris le serveur. Cependant, dans le serveur il existe un autre fichier qui doit contenir ce bloc, le fichier /etc/perfsonar/meshconfig-guiagent.conf. Je ne savais pas qu'il devait le contenir. Enfin, j'ai ajouté ce bloc et tout a bien marché.

4.4 Automatisation de l'administration de nœuds

Cette partie décrit une étape importante de ce qui j'ai apporté à l'observatoire pendant ces trois mois de stage. J'ai proposé la création d'un moyen (un script) pour administrer le fichier meshconfig et faciliter sa modification dans l'avenir.

D'abord, j'explique le besoin qui m'a amené à partir sur cette idée et les alternatives possibles. Ensuite, je décris les éléments du script, sa structure et son fonctionnement. Finalement, je propose quelques améliorations envisagées.

4.4.1 Analyse du besoin

Le fichier meshconfig est très important pour toute l'infrastructure perfSONAR à l'observatoire. Il faut du temps pour bien le construire. Son administration peut être lourde en la faisant à la main. Il y a une syntaxe spécifique à respecter et au début il peut y avoir trop d'erreurs encore de syntaxe ou de frappe puisque la maîtrise n'est pas tout à fait bien acquise. En plus, il faut récréer une fichier JSON, redémarrer les services, etc. Tout cela avec des erreurs peut prendre beaucoup trop de temps même si on voulait juste ajouter un nouveau nœud.

Ensuite, vu que la mise en place de perfSONAR à l'observatoire est récente, elle est censée changer avec le temps, surtout par rapport à l'ajout ou suppression de nœuds, c'est le cas des miroirs et d'autres nœuds de test. C'est pourquoi on a décidé de développer un outil permettant de le faire plus rapidement et en évitant les erreurs de syntaxe ou de frappe.

4.4.2 Alternatives possibles au script

Si on parle plus précisément de l'ajout et suppression de nœuds, perfSONAR fournit un outil appelé « *Automatic Test Configuration with MeshConfig* ». Cet outil est prévu pour les grandes « mailles » ou pour des mailles changeant très fréquemment. Il existe deux manières de le mettre en place :

1. Pour tester un groupe fixe de nœuds : ici, tous les nouveaux testeurs (les nœuds qui vont être ajoutés automatiquement) feront des tests vers un ensemble de nœuds fixes (définis explicitement

dans le fichier meshconfig). Les nouveaux testeurs n'auront pas de connaissance sur les autres testeurs dynamiques et pour le coup ils ne pourront pas faire des tests vers eux. En plus, ces deniers n'apparaîtront pas dans le tableau de bord de MaDDash.

2. Pour tester un groupe dynamique de nœuds : ici, tous les testeurs dynamiques ont la capacité de détecter la présence d'autres testeurs ce qui permet de lancer des tests en maille. Pour cela, il faut créer un *Private Lookup Service*[8].

Analysons les deux options. La première peut être un bon choix parce que l'observatoire a besoin de faire des tests depuis les miroirs vers un seul nœud fixe et à l'inverse, celui du bâtiment Sud qui est le nœud principal, n'a pas besoin de lancer de tests entre les miroirs. Cependant, avec cette option on perd l'affichage des miroirs dans le tableau de bord.

Ensuite, avec la deuxième option on conserve les nouveaux testeurs dans le tableau de bord mais les tests seront lancés aussi entre les miroirs, une chose à éviter pour l'observatoire.

Enfin, on a décidé de créer un outil d'administration pour les nœuds et les tests.

4.4.3 Le script

Description et fonctions

Le script permet de récréer un fichier *meshconfig* prêt à être converti en format JSON et à être utilisé par les services de perfSONAR. Ses fonctions sont :

- Lister les nœuds
- Ajouter des nœuds
- Supprimer des nœuds
- Choisir les tests à lancer par chaque groupe
- Modifier les paramètres de chaque test
- Afficher le fichier meshconfig actuel

Même si jusqu'à cette partie on n'avait aucune sonde à l'extérieur, on savait que l'on en aurait à l'avenir, c'est pourquoi le script a pris en compte cette partie.

Une arborescence requise

Le premier réflexe pour stocker et lire les informations des nœuds (description, ip), des groupes (nom, type, membres), des tests (nom, paramètres) et les relations entre tous ces éléments a été une base de données. Cependant, il fallait administrer un service en plus, faire la maintenance en cas de panne et

8. Outil permettant de rechercher des services, nœuds ou interfaces perfSONAR

cette base de données allait prendre des ressources du serveur perfSONAR (si on y l'installait) ou sur une autre machine (virtuelle ou physique). De plus, la quantité de données et leurs relations n'étaient pas nombreuses. Enfin, le script n'est pas un outil qui sera utilisé tous les jours et avoir une base de données tournant juste pour cet outil n'est pas très efficace.

J'ai donc décidé de traiter tous les éléments du script à l'aide d'une arborescence de fichiers. Cette dernière est composée des répertoires :

a) backup : à chaque lancement du script, une copie du fichier meshconfig actuel sera sauvegardé dans ce répertoire. Si une erreur apparaît pendant la création du nouveau fichier JSON, la copie sera restaurée.

b) sites : il héberge tous les blocs de configuration de chaque nœud, chacun dans un fichier différent

c) groupes : il contient deux sous-répertoires

- mesh : hébergeant les membres du groupe de nœuds internes de l'observatoire
- disjoint : hébergeant les membres du groupe de nœuds à l'extérieur. Un peu particulier parce qu'il contient encore deux sous-répertoires. L'un pour les membres du type 'a' et l'autre pour les membres du type 'b'

d) tests : il contient un fichier pour chaque test existant. Ces fichiers ont été crées précédemment et on peut y en rajouter d'autres valides bien sûr. Il contient aussi un sous-répertoire appelé 'actives' avec un fichier 'actives.cfg' qui décrit quels tests appartiennent à chaque groupe

e) le fichier meshconfig : le fichier de configuration résultant

Structure

L'outil comporte un script principal codé en Bash, deux scripts codés en Python chargés de créer les fichiers de configuration et deux templates de configuration dont les scripts Python vont se servir.

Le script principal appelé *manage_sondes.sh* (voir Annexe 6.2, page 47) est chargé de contrôler toute l'exécution. Il fait les appels aux scripts Python, gère les erreurs, gère ses propres dépendances, demande les informations nécessaires et crée une interface graphique à l'aide de *Whiptail*[9].

Le script *sonde_conf.py* (voir Annexe 6.4, page 64) se sert de deux templates en format *Jinja2*[10] pour créer une configuration partielle du fichier meshconfig. Le premier template appelé *sonde_obas.temp* (voir Annexe 6.5, page 66) comporte le bloc 'site' qui correspond à la configuration d'un nouveau nœud simple qui appartient à l'observatoire. Si un noeud appartient à une organisation différente, le template *no_agent.temp* (voir Annexe 6.6, page 67) est utilisé.

9. Outil permettant de créer des interfaces graphiques en shell
10. Moteur de templates pour le langage de programmation Python.

Fonctionnement général

Le principe du script est de créer tout d'abord le bloc de configuration du nœud à l'aide des templates et le placer dans le répertoire 'sites'. Ensuite, le script crée un lien symbolique vers le répertoire 'groupes' auquel correspond le nœud pour les « relier ». Puis, un script Python parcours toute l'arborescence pour récupérer la configuration de chaque nœud, les groupes, les blocs de tests et les relations entre les groupes et les tests et crée un nouveau fichier meshconfig et en génère le fichier JSON. À la fin, les services perfSONAR concernés sont redémarrés pour prendre en compte les modifications. Ci-dessous un petit schéma de son fonctionnement :

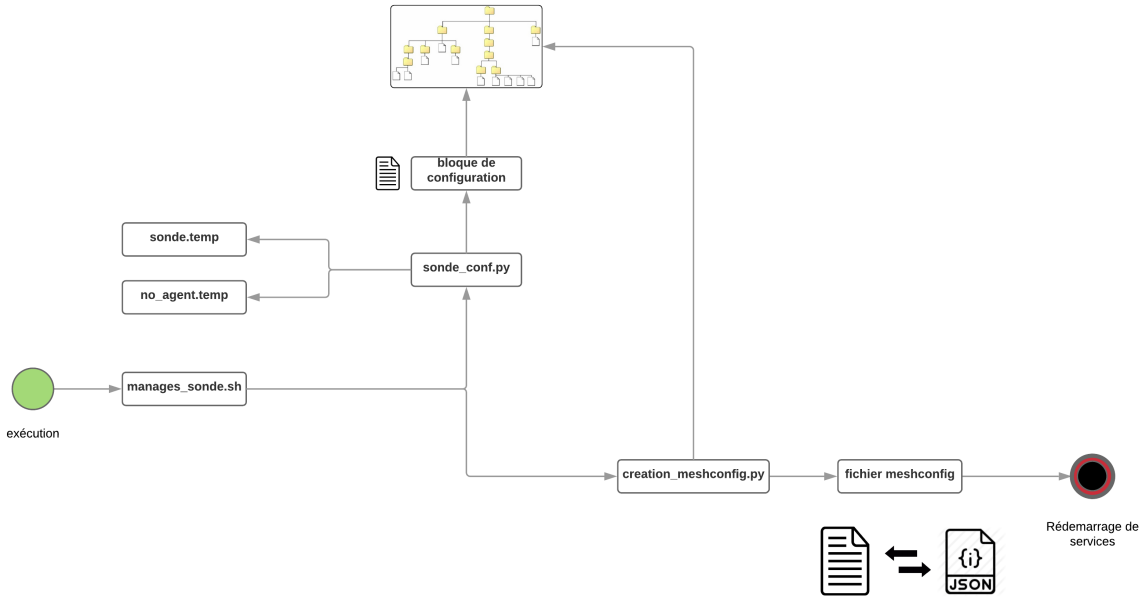


FIGURE 11 – Fonctionnement du script

4.4.4 Utilisation

Dans cette section je me permets d'expliquer l'utilisation du script et d'inclure quelques captures d'écran parce que j'aimerais montrer à quoi ressemble cet outil une fois fini.

Tâche list

La tache 'list' affiche la liste de tous les nœuds configurés.

```
./manage_sondes.sh --action=list --dir=/home/psonde/mesh
```

```
SCRIPT POUR L'ADMINISTRATION DES SONDES perfSONAR À L'OBSERVATOIRE ...

          HOST                          DESCRIPTION

[ 1 ]   ops.u-strasbg.fr              Sonde Batiment Sud
[ 2 ]   ops-se.u-strasbg.fr           Sonde Batiment Est
[ 3 ]   ops-sc.u-strasbg.fr           Sonde Batiment Coupole
[ 4 ]   sbgperfps1.in2p3.fr           Sonde IPHC
[ 5 ]   193.55.200.68                 Sonde Renater Paris
[ 6 ]   206.12.59.6                   Sonde CANADA
```

FIGURE 12 – Tâche list

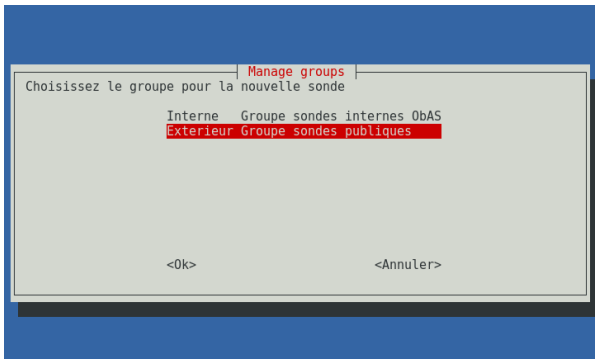
Tâche add

La tâche 'add' demande d'abord si le nœud à ajouter appartient à l'observatoire pour décider quel template utiliser.

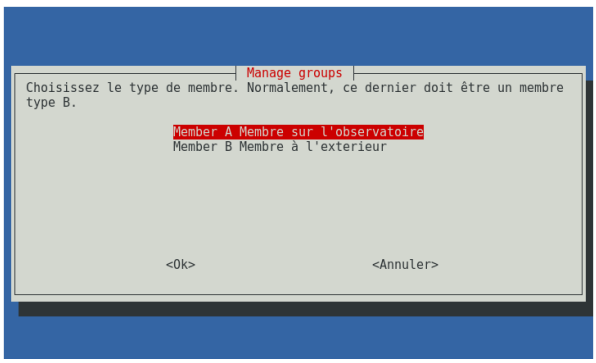
```
./manage_sondes.sh --action=add --dir=/home/psonde/mesh
```

Si le choix est 'Observatoire' il faut entrer une description et ensuite l'adresse IP. Sinon, si le choix est 'Autre', il faut entrer le nom de l'organisation, une description et ensuite l'adresse IP.

Puis, il faut choisir le groupe (Figure 13a), soit 'Interne' ou 'Extérieur' (si c'est extérieur il faut choisir 'Membre A' ou 'Membre B' (Figure 13b), normalement celui doit être 'Membre B' parce que le noeud Sud central est le 'Membre A').



(a) Type de noeud



(b) Sélection du groupe

FIGURE 13 – Sélection des tests par groupe

Tâche supprimer

La tâche supprimer affiche d'abord un RadioList [11] contenant tous les nœuds tournant. Il faut en choisir un pour continuer avec la suppression. Ensuite un message de confirmation est affiché et enfin le nœud est supprimé.

```
./manage_sondes.sh --action=delete --dir=/home/psonde/mesh
```

FIGURE 14 – Sélection du nœud à supprimer

Tâche conftest

Cette tâche concerne à l'administration des tests. Elle gère les tests qui vont se lancer par chaque groupe et leur paramètres.

```
./manage_sondes.sh --action=conftest --dir=/home/psonde/mesh
```

Premièrement, un message est affiché. Il demande si on veut modifier les paramètres d'un test ou choisir les tests à lancer par chaque groupe.

Si 'Suivant' est choisi, le script montre les tests qui tournent à ce moment là pour les deux groupes de l'observatoire (Figure 15, page 33). On a la possibilité de sélectionner ceux que l'on veut pour le groupe 'Interne' et pour le groupe 'Extérieur' grâce à un CheckList [12].

11. Élément de Whiptail permettant de choisir un seul élément d'une liste
12. Élément de Whiptail permettant de choisir plusieurs éléments d'une liste

(a) Groupe interne (b) Groupe extérieur

FIGURE 15 – Sélection des tests par groupe

Chaque test défini doit avoir une description. Cette dernière est celle affichée dans le tableau de bord de MaDDash. Si de nouveaux tests sont sélectionnés, le script demande d'entrer une description. Par ailleurs, il conserve les descriptions de tous les tests qui n'ont pas changés.

Ensuite, si 'Avancé' est choisi. Le script affiche un RadioList avec les tests disponibles pour en choisir un à modifier.

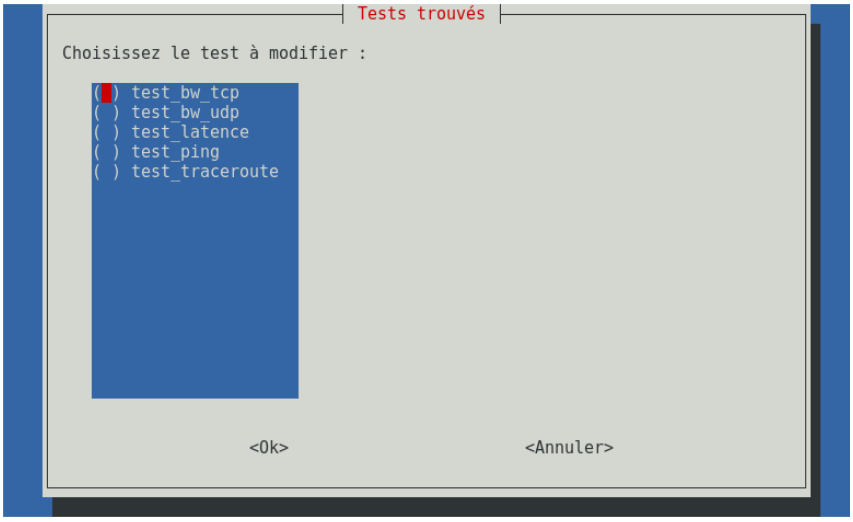


FIGURE 16 – Choisir un test à modifier

Lorsqu'un test est choisi, la liste de leurs paramètres est affichée. Par exemple, un test de latence a les paramètres suivants :

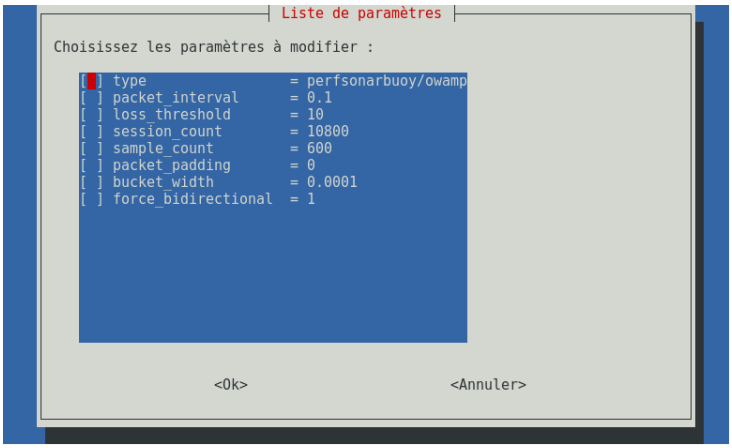
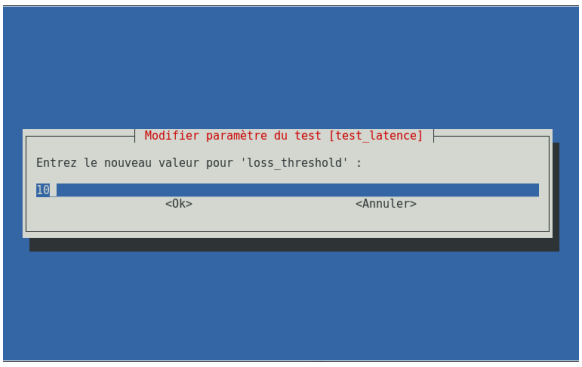
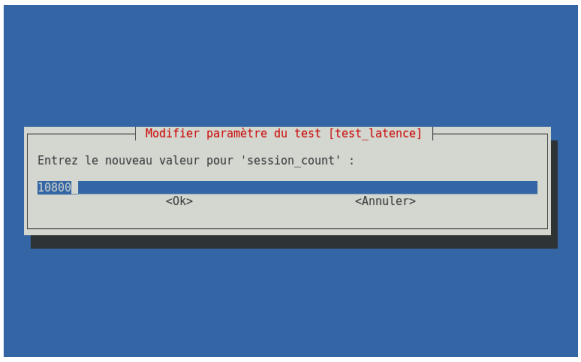


FIGURE 17 – Exemple des paramètres d'un test

Il s'agit d'un CheckList permettant de choisir plusieurs paramètres à modifier. Si on en choisit plusieurs, le script demandera une nouvelle valeur autant de fois que de paramètres sélectionnés (Figure 18. Une avantage est que à chaque fois que le script demande une nouvelle valeur, il met par défaut la valeur actuelle dans la boîte de texte.



(a) Paramètre 'loss_threshold'



(b) Paramètre 'session_count'

FIGURE 18 – Modification de valeurs

Tâche afficher le fichier meshconfig

Enfin, la possibilité d'afficher le fichier résultant à n'importe quel moment est offerte par cette tâche.

```
./manage_sondes.sh --action=meshfile --dir=/home/psonde/mesh
```

4.5 Sondes à l'extérieur

Sans doute, la supervision du débit et de la latence vue depuis l'extérieur est plus intéressante. C'est pourquoi la deuxième partie de l'infrastructure est l'ajout de nœuds placés physiquement loin de l'observatoire (Figure 19). Les premiers nœuds pensés sont :

1. Renater Paris : ce nœud appartient à RENATER. Il n'est pas administré pour l'observatoire mais on a la permission pour l'interroger sans aucun problème. En plus, on peut contacter ses administrateurs en cas de soucis.

2. CADC [13] à Victoria au Canada : ce nœud appartient à l'observatoire et récupère notre fichier de configuration centralisée

3. IPHC Strasbourg : ce nœud est administré par l'IPHC [14] de Strasbourg et il est public pour l'interroger.

4. Miroirs de l'observatoire : nœuds à rajouter possiblement dans le futur. Chaque service du CDS a plusieurs miroirs répartis dans le monde entier mais il faut avoir une approbation pour pouvoir mettre en place perfSONAR sur chacun d'eux. Pour le moment ils ne figureront pas dans le tableau de bord.

FIGURE 19 – Emplacement nœuds externes

Il a fallu donc créer deux groupes pour construire le fichier de configuration final (voir Annexe 6.1, page 40), l'un pour les nœuds en interne et l'autre pour ceux à l'extérieur. Cela est la différence la plus grande par rapport aux fichiers créés dans les étapes précédentes. Ce qui est nouveau est le groupe disjoint et ses membres 'b'. Certains sont des nœuds dont on n'a pas la maîtrise et ils appartiennent à une autre organisation. Pour des nœuds de ce type perfSONAR a une instruction appelée no_agent, cela permet de

13. Canadian Astronomy Data Centre
14. Institut Pluridisciplinaire Hubert CURIEN

spécifier que ce nœud ne lit pas notre configuration et en conséquence c'est à nous de stocker les résultats des tests vers lui et à l'inverse.

Test d'un nœud chez Thomas

L'un des premiers tests de nœuds à l'extérieur a été l'installation de perfSONAR sur une machine chez mon tuteur Thomas. Il a ouvert tous les ports requis du pare-feu de l'observatoire et de son routeur. À l'aide du script, j'ai ajouté ce nœud à la configuration. Ensuite, j'ai vérifié que les tests se lançaient correctement. Cependant, cela n'a pas marché du tout à cause d'un problème avec NAT. Quand un client est derrière de NAT, l'adresse source passée n'est pas la même que l'adresse actuelle vue par le serveur[linlog, 2018]. En effet, l'adresse source change lors du passage par NAT. Par ailleurs, lors d'une requête de session, le serveur OWAMP vérifie que ces deux adresses sont identiques afin d'éviter des attaques. On peut bien contourner cette vérification mais juste lorsque l'on fait de tests à la main.

Enfin, cette sonde était de test et elle nous a servi pour une phase d'apprentissage et il ne valait pas la peine de rester longtemps à la déboguer ou à essayer de la mettre en place.

Miroirs

Le service informatique avait prévu le déploiement de perfSONAR sur chacun des miroirs de l'observatoire des services du CDS. Cependant, il faut avoir l'autorisation de le faire et on a eu des contraintes liées à la sécurité réseau.

Renater Paris

On a à notre disposition un nœud à Paris administré par RENATER et aussi le support de ses administrateurs système en cas de contraintes. Au début je n'arrivais pas à établir les connexions avec ce nœud, j'ai donc contacté M. Renan Philippe, l'un des administrateurs pour vérifier si c'était un problème lié à leur pare-feu. Après quelques échanges il m'a proposé de demander le support d'Osiris parce qu'il semblait qu'ils n'avaient pas la route vers l'adresse. Mon co-encadrant M. Saillard Christophe a pris contact avec eux et en effet ils n'avaient pas la route, ils l'ont ajoutée et on est immédiatement arrivé à lancer les tests.

Nœud Victoria, Canada

Ce nœud est le seul à lire notre configuration centralisée. Il est placé dans le CADC à Victoria, Canada où on a l'aide d'un administrateur système, M. Ouellette John qui a créé une machine virtuelle sur un cluster OpenStack.

D'autres nœuds

Par ailleurs, j'ai inclus un autre nœud à notre configuration, celui du IPHC de Strasbourg. Celui est un noeud publique qui permet de faire plusieurs tests. Je l'ai retrouvé grâce au site web de perfSONAR Lookup Service Directory. D'ailleurs, celui a été le premier noeud à l'extérieur que j'ai ajouté à notre fichier meshconfig pour tester la connectivité une fois on a ouvert les ports du pare-feu.

Actuellement, il y a plus de 2000 nœuds perfSONAR déployées partout et la plupart peuvent être interrogés par d'autres nœuds. Cela est une grande avantage parce que l'on peut les rajouter pour nos tests.

5 Conclusion

Ce stage m'a permis d'apprendre de nouvelles choses, tant sur la partie professionnelle que personnelle. Cela a été une expérience très enrichissante du point de vue des compétences.

Premièrement, j'ai eu l'opportunité de connaître de nouvelles méthodologies de travail et d'étude. J'ai appris à utiliser d'autres technologies très différentes de celles utilisées dans mes cours à l'IUT ou dans mon pays. J'ai pu aussi mettre en pratique toutes les connaissances acquises dans mes cours.

Par ailleurs, mon projet professionnel est de devenir administrateur système et réseau, ce stage m'a beaucoup apporté dans ce domaine. J'ai eu l'opportunité d'échanger avec d'autres administrateurs systèmes et réseau soit par mail, téléphone ou même par vidéo-conférence. Par exemple, mes encadrants et moi, nous avons eu une vidéo-conférence en anglais avec un membre de notre équipe perfSONAR à Victoria au Canada et une autre en français avec un informaticien chez Renater. Puis, avec le déploiement de perfSONAR j'ai du relever certains challenges liés surtout à la coordination entre les membres de ce déploiement. D'abord, parce qu'il y avait des sondes que l'on ne maîtrisait pas directement et parfois leurs administrateurs n'avaient pas le contrôle total sur leur infrastructure réseau. De ce fait, le débogage se rendait plus difficile. D'un autre côté, j'ai du travailler avec des outils d'administration et supervision réseau comme iperf, owamp ou bwctl et développer en plus mes compétences en scripting et troubleshooting, des parties fondamentales pour tout administrateur système. De plus, j'ai découvert que l'automatisation est un sujet que m'intéresse énormément et dans lequel que je voudrais continuer à m'investir.

En ce qui concerne ma vie personnelle, j'ai renforcé surtout mon autonomie et organisation. J'ai appris à mieux gérer mon temps et à bien planifier tous mes travaux. Ensuite, cela a été aussi l'occasion de pratiquer mon français et mon anglais à l'oral et à l'écrit et de m'adapter à une autre culture de travail et de relations interpersonnelles.

Cette expérience a été très fructueuse pour moi, elle va me permettre très sûrement de m'insérer plus efficacement dans le monde de travail en France ou dans mon pays d'origine.

Enfin, à l'issue de la période définie de mon stage, mes encadrants m'ont proposés de la prolonger jusqu'au 6 juillet pour travailler sur la mise en œuvre de la solution de supervision Zabbix. Je suis donc très heureux de pouvoir continuer à travailler au sein de l'Observatoire Astronomique de Strasbourg.

Bibliographie

[esnet, 2018a] ESNET (2018a). Documentation officielle Esmond. https://software.es.net/esmond. Consulté le 27 mai 2018.

[esnet, 2018b] ESNET (2018b). MaDDash : Monitoring and Debugging Dashboard. https://software.es.net/maddash/. Consulté le 1 juin 2018.

[linlog, 2018] LINLOG (2018). OWAMP problems behind NATowamp problems behind nat. http://linlog.blogspot.com/2009/06/owamp-problems-behind-nat.html. Consulté le 5 juin 2018.

[LinuxContainers, 2018] LINUXCONTAINERS (2018). Linux containers : LXD. https://linuxcontainers.org/fr/. Consulté le 28 mai 2018.

[observatoire astronomique, 2018] observatoire ASTRONOMIQUE (2018). Site officiel : Observatoire astronomique de Strasbourg. https://astro.unistra.fr. Consulté le 20 mai 2018.

[perfSONAR, 2018a] PERFSONAR (2018a). perfSONAR : Firewall and Security Software. https://docs.perfsonar.net/manage_security.html. Consulté le 1 juin 2018.

[perfSONAR, 2018b] PERFSONAR (2018b). perfSONAR : Monitoring architectural examples. https://www.perfsonar.net/deploy/deployment-use-cases/. Consulté le 1 juin 2018.

[perfSONAR, 2018c] PERFSONAR (2018c). perfSONAR : What is perfsonar ? http://docs.perfsonar.net/intro_about.html. Consulté le 25 mai 2018.

[Wikipedia, 2018a] WIKIPEDIA (2018a). Wikipedia : LXC. https://fr.wikipedia.org/wiki/LXC. Consulté le 28 mai 2018.

[Wikipedia, 2018b] WIKIPEDIA (2018b). Wikipedia : Network Time Protocole. https://fr.wikipedia.org/wiki/Network_Time_Protocol. Consulté le 25 mai 2018.

[Wikipedia, 2018c] WIKIPEDIA (2018c). Wikipedia : VMware VSphere. https://fr.wikipedia.org/wiki/VMware_vSphere. Consulté le 5 juin 2018.

6 Annexes

6.1 Fichier meshconfig.conf final

```
description Sondes PerfSONAR Observatoire

<organization>
    description     Observatoire Astronomique de Strasbourg


    <site>
        <host>
              description Sonde Batiment Sud
              address ops.u-strasbg.fr

              <measurement_archive>
                    type traceroute
                    read_url http://ops.u-strasbg.fr/esmond/perfsonar/archive
                    write_url http://ops.u-strasbg.fr/esmond/perfsonar/archive
              </measurement_archive>

              <measurement_archive>
                    type perfsonarbuoy/owamp
                    read_url http://ops.u-strasbg.fr/esmond/perfsonar/archive
                    write_url http://ops.u-strasbg.fr/esmond/perfsonar/archive
              </measurement_archive>

              <measurement_archive>
                    type perfsonarbuoy/bwctl
                    read_url http://ops.u-strasbg.fr/esmond/perfsonar/archive
                    write_url http://ops.u-strasbg.fr/esmond/perfsonar/archive
              </measurement_archive>

              <measurement_archive>
                    type pinger
                    read_url http://ops.u-strasbg.fr/esmond/perfsonar/archive
                    write_url http://ops.u-strasbg.fr/esmond/perfsonar/archive
              </measurement_archive>
        </host>
```

```
36      </site>
37
38      <site>
39          <host>
40                description Sonde Batiment Est
41                address ops-se.u-strasbg.fr
42
43                <measurement_archive>
44                      type traceroute
45                      read_url http://ops-se.u-strasbg.fr/esmond/perfsonar/archive
46                      write_url http://ops-se.u-strasbg.fr/esmond/perfsonar/
                            archive
47                </measurement_archive>
48
49                <measurement_archive>
50                      type perfsonarbuoy/owamp
51                      read_url http://ops-se.u-strasbg.fr/esmond/perfsonar/archive
52                      write_url http://ops-se.u-strasbg.fr/esmond/perfsonar/
                            archive
53                </measurement_archive>
54
55                <measurement_archive>
56                      type perfsonarbuoy/bwctl
57                      read_url http://ops-se.u-strasbg.fr/esmond/perfsonar/archive
58                      write_url http://ops-se.u-strasbg.fr/esmond/perfsonar/
                            archive
59                </measurement_archive>
60
61                <measurement_archive>
62                      type pinger
63                      read_url http://ops-se.u-strasbg.fr/esmond/perfsonar/archive
64                      write_url http://ops-se.u-strasbg.fr/esmond/perfsonar/
                            archive
65                </measurement_archive>
66          </host>
67      </site>
68
69      <site>
70          <host>
71                description Sonde Batiment Coupole
```

```
72                address ops-sc.u-strasbg.fr
73
74                <measurement_archive>
75                      type traceroute
76                      read_url http://ops-sc.u-strasbg.fr/esmond/perfsonar/archive
77                      write_url http://ops-sc.u-strasbg.fr/esmond/perfsonar/
                            archive
78                </measurement_archive>
79
80                <measurement_archive>
81                      type perfsonarbuoy/owamp
82                      read_url http://ops-sc.u-strasbg.fr/esmond/perfsonar/archive
83                      write_url http://ops-sc.u-strasbg.fr/esmond/perfsonar/
                            archive
84                </measurement_archive>
85
86                <measurement_archive>
87                      type perfsonarbuoy/bwctl
88                      read_url http://ops-sc.u-strasbg.fr/esmond/perfsonar/archive
89                      write_url http://ops-sc.u-strasbg.fr/esmond/perfsonar/
                            archive
90                </measurement_archive>
91
92                <measurement_archive>
93                      type pinger
94                      read_url http://ops-sc.u-strasbg.fr/esmond/perfsonar/archive
95                      write_url http://ops-sc.u-strasbg.fr/esmond/perfsonar/
                            archive
96                </measurement_archive>
97          </host>
98      </site>
99
100     <site>
101         <host>
102               description Sonde CANADA
103               address 206.12.59.6
104
105               <measurement_archive>
106                     type traceroute
107                     read_url http://206.12.59.6/esmond/perfsonar/archive
```

```
                write_url http://206.12.59.6/esmond/perfsonar/archive
        </measurement_archive>

        <measurement_archive>
                type perfsonarbuoy/owamp
                read_url http://206.12.59.6/esmond/perfsonar/archive
                write_url http://206.12.59.6/esmond/perfsonar/archive
        </measurement_archive>

        <measurement_archive>
                type perfsonarbuoy/bwctl
                read_url http://206.12.59.6/esmond/perfsonar/archive
                write_url http://206.12.59.6/esmond/perfsonar/archive
        </measurement_archive>

        <measurement_archive>
                type pinger
                read_url http://206.12.59.6/esmond/perfsonar/archive
                write_url http://206.12.59.6/esmond/perfsonar/archive
        </measurement_archive>
    </host>
  </site>
</organization>


<organization>
  description  IPHC Strasbourg

  <site>
    <host>
        no_agent 1
        description Sonde IPHC
        address sbgperfps1.in2p3.fr
    </host>
  </site>
</organization>


<organization>
  description  RENATER
```

```
148
149      <site>
150          <host>
151                no_agent 1
152                description Sonde Renater Paris
153                address 193.55.200.68
154          </host>
155      </site>
156  </organization>
157
158  <test_spec test_bw_udp>
159     type perfsonarbuoy/bwctl
160     tool bwctl/iperf3
161     protocol udp
162     interval 86400
163     duration 10
164  </test_spec>
165
166  <test_spec test_bw_tcp>
167     type perfsonarbuoy/bwctl
168     tool bwctl/iperf3
169     protocol tcp
170     interval 86400
171     duration 20
172     force_bidirectional 1
173     random_start_percentage 10
174     omit_interval 5
175  </test_spec>
176
177  <test_spec test_latence>
178     type perfsonarbuoy/owamp
179     packet_interval 0.1
180     loss_threshold 10
181     session_count 10800
182     sample_count 600
183     packet_padding 0
184     bucket_width 0.0001
185     force_bidirectional 1
186  </test_spec>
187
```

```
<test_spec test_ping>
   type pinger
   test_interval 300
   packet_count 10
   packet_interval 1
   packet_size 1000
</test_spec>

<test_spec test_traceroute>
   type traceroute
   test_interval 600
   packet_size 40
</test_spec>

<group obas_interne_mesh>
   type mesh

   member ops.u-strasbg.fr
   member ops-se.u-strasbg.fr
   member ops-sc.u-strasbg.fr
</group>

<group obas_exterieur_disj>
   type disjoint

   a_member ops.u-strasbg.fr

   b_member sbgperfps1.in2p3.fr
   b_member 193.55.200.68
   b_member 206.12.59.6

</group>

<test>
   description    Test Ping Sondes Internes
   group    obas_interne_mesh
   test_spec    test_ping
</test>

<test>
```

```
228     description    Test Bande Passante UDP Sondes Internes
229     group    obas_interne_mesh
230     test_spec    test_bw_udp
231   </test>
232
233   <test>
234     description    Test Bande Passante TCP Sondes Internes
235     group    obas_interne_mesh
236     test_spec    test_bw_tcp
237   </test>
238
239   <test>
240     description    Test Traceroute Interne
241     group    obas_interne_mesh
242     test_spec    test_traceroute
243   </test>
244
245   <test>
246     description    Test Latence Sondes Internes
247     group    obas_interne_mesh
248     test_spec    test_latence
249   </test>
250
251   <test>
252     description    Test Ping Sondes Exterieur
253     group    obas_exterieur_disj
254     test_spec    test_ping
255   </test>
256
257   <test>
258     description    Test Bande Passante TCP Sondes Exterieur
259     group    obas_exterieur_disj
260     test_spec    test_bw_tcp
261   </test>
262
263   <test>
264     description    Test Traceroute Sondes Exterieur
265     group    obas_exterieur_disj
266     test_spec    test_traceroute
267   </test>
```

```
268
269 <test>
270    description    Test Latence Sondes Exterieur
271    group    obas_exterieur_disj
272    test_spec    test_latence
273 </test>
```

6.2 Script manages_sondes.sh

```
1  #!/bin/bash
2
3  # Ce script est le script principal, ici on valide toutes les entrées, les
       erreurs qui peuvent apparaître, les dépendances et les appels aux scripts
       Python
4
5  ACTION="" ; DIR="" ; path_SRV="/var/www/html" ; jour=$(date "+%d-%m-%Y")
6  heure=$(date "+%H:%M") ; nomBack="meshconfig-$jour-$heure.bak"
7  TYPE="" ; MEMBRE="" ; no_agent=0
8  bold=$(tput bold) ; normal=$(tput sgr0) ; italic=$(tput sitm)
9  under=$(tput smul) ; tests_mesh="" ; tests_disj="" ; sondes=
10
11 arborescence () {
12   echo """
13 EXAMPLE DES FICHIERS REQUIS PAR LE SCRIPT ET DE L'ARBORESCENCE DU REPERTOIRE
14
15 """
16 }
17
18 aide()
19 {
20    #Usage
21 }
22
23 #Message appelé lors de chaque erreur
24 die () {
25    echo -e "\n
          +----------------------------------------------------------------+\n"
26    echo -e "Le script à echoué à cause de l'erreur suivante : \n$1\n"
```

```
27      echo -e "\n
            +-----------------------------------------------------------------+\n"
28      if [[ $2 == "5" ]] ; then arborescence ; fi
29      exit "$2"
30  }
31
32  #D'abord assurer que le script est lancé en tant que superutilisateur
33  assurer_root () {
34      if [ "$EUID" -ne "0" ] ; then
35          return 1
36      fi
37      return 0
38  }
39
40  #Le script a besoin de certains paquets pour fonctionner correctement. C'est le
        cas de 'whiptail' et 'python-yaml'. Le premier sert à créer une interface
        graphique en terminal, utilisé pour demander les informations nécessaires
        pour la création de la nouvelle sonde. Le paquet 'python-yaml' est né
        cessaire pour pouvoir traiter des fichiers .yaml, les importer dans et avec
        un template jinja2 créer un nouveau fichier de configuration.
41
42  dependences_script () {
43      if [[ -e "/etc/debian_version" ]]; then
44          if ! command -v whiptail>/dev/null 2>&1 ; then apt-get -y install whiptail
                ; fi
45      elif [[ -e "/etc/centos-release" ]]; then
46          if ! command -v whiptail>/dev/null 2>&1 ; then yum -y install newt; fi
47          if [ -z $(rpm -qa | grep yaml) ] ; then yum -y install python-yaml; fi
48      else
49          echo "Ce système n'est pas supporté. Pour le moment, le script est fait
                pour Debian 8/9 et CentOS 6/7"
50      fi
51      return 0
52  }
53
54  #Il faut vérifier que le script a les paramètres requis
55  assurer_entres () {
56      if [[ $ACTION == "" ]] || [[ $DIR == "" ]] ; then
57          return 1
58      fi
```

```
59
60      return 0
61  }
62
63  #Vérifier que tous les fichiers qui comportent le script sont dans les ré
        pertoire courant
64  fichiers_script_presents () {
65      if [ ! -f "./creation_meshconfig.py" ] || [ ! -f "./sonde_obas.temp" ] || [ !
            -f "./sonde_conf.py" ] ; then
66          return 1
67      fi
68      if [ ! -d "$DIR/sites" ] || [ ! -d "$DIR/backup" ] || [ ! -d "$DIR/groupes" ]
            || [ ! -d "$DIR/tests" ] || [ ! -f "$DIR/meshconfig.conf" ]  ; then
69          return 1
70      fi
71
72      return 0
73  }
74
75  lister_tests () {
76    for file in $(ls -p $DIR/tests/ | grep -v /) ; do
77      tests[i]=$(echo ${file%.*}) ; (( i++ ))
78      tests[i]="" ; (( i++ ))
79      tests[i]="OFF" ; (( i++ ))
80    done
81  }
82
83
84  #Cette function permet de activer les cases du checklist selon le fichier
        actives.cfg
85  sel_items_checklist () {
86    declare -a listeArg=("${!1}")
87
88    lister_tests
89
90    for i in $(seq 0 ${#tests[@]}) ; do
91          for val in "${listeArg[@]}" ; do
92              if [[ ${tests[i]} == $val ]] ; then
93                  ac=$i+2
94                  tests[ac]="ON"
```

```
95              fi
96          done
97      done
98  }
99
100 active_tests () {
101     i=0 ; ind=0
102     while IFS='' read -r linea || [[ -n "$linea" ]]; do
103         div=(${linea//,/ })
104         if [[ ${div[0]} == "obas_interne_mesh" ]] ; then
105             mesh[i]=${div[1]}
106             (( i++ ))
107         else
108             disj[ind]=${div[1]}
109             (( ind++ ))
110         fi
111     done < $DIR/tests/actives/actives.cfg
112     i=0
113     sel_items_checklist mesh[@]
114
115     tests_mesh=$(whiptail --title "Tests Groupe INTERNE - MESH" --checklist --
        separate-output "\nLes tests qui tournent actuellement pour ce groupe sont
         ceux déjà selectionés. Activez/desactivez ceux qui vous préferez." 25 78
        16 "${tests[@]}" 3>&1 1>&2 2>&3)
116     if [ $? = 1 ] || [[ $tests_mesh == "" ]] ; then die "Tache interrompue." 1 ;
        fi
117
118     i=0
119     sel_items_checklist disj[@]
120
121     tests_disj=$(whiptail --title "Tests Groupe EXTERIEUR - DISJOINT" --checklist
        --separate-output "\nLes tests qui tournent actuellement pour ce groupe
        sont ceux déjà selectionés. Activez/desactivez ceux qui vous préferez." 25
         78 16 "${tests[@]}" 3>&1 1>&2 2>&3)
122     if [ $? = 1 ] || [[ $tests_disj == "" ]] ; then die "Tache interrompue." 1 ;
        fi
123
124 }
125
126 #Demander les informations concernant la sonde
```

```
127 information () {
128    if (whiptail --title "Type de sonde" --yesno --yes-button "Observatoire" --no
           -button "Autre" "La sonde à ajouter s'agit d'une sonde qui appartient à l
           'observatoire (que l'on maitrise) ou d'une sonde d'une autre organisation
            (ex. celle de RENATER) ?" 12 78) then
129       no_agent=0
130    else
131       org=$(whiptail --inputbox "Entrez le nom de l'organisation." 8 78 --title
              "Information" 3>&1 1>&2 2>&3)
132       no_agent=1
133    fi
134    descr=$(whiptail --inputbox "Entrez une description pour la sonde." 8 78 --
           title "Information" 3>&1 1>&2 2>&3)
135
136    exitstatus=$?
137    if [ $exitstatus = 0 ]; then
138       echo "Description établie."
139    else
140       echo "Installation interrompue."
141       return 1
142    fi
143
144    addr=$(whiptail --inputbox "Entrez l'addresse de la sonde." 8 78 --title "
           Information" 3>&1 1>&2 2>&3)
145
146    exitstatus=$?
147    if [ $exitstatus = 0 ]; then
148       echo "Addresse établie."
149    else
150       echo "Installation interrompue."
151       return 1
152    fi
153
154    groupe=$(whiptail --title "Manage groups" --menu "Choisissez le groupe pour
           la nouvelle sonde" 16 78 5 \
155         "Interne" "Groupe sondes internes ObAS"\
156         "Exterieur" "Groupe sondes publiques" 3>&2 2>&1 1>&3)
157
158    opt=$(echo $groupe | tr '[:upper:]' '[:lower:]')
159    if [ $? != 0 ] ; then return 1; fi
```

```
160
161
162     if [[ $opt == "interne" ]] ; then
163        TYPE="mesh"
164     elif [[ $opt == "exterieur" ]] ; then
165        TYPE="disjoint"
166
167        tipo=$(whiptail --title "Manage groups" --menu "Choisissez le type de
               membre. Normalement, ce dernier doit être un membre type B." 16 78 5 \
168                 "Member A" "Membre sur l'observatoire"\
169                 "Member B" "Membre à l'exterieur" 3>&2 2>&1 1>&3)
170        if [ $? != 0 ] ; then return 1; fi
171
172        opt2=$(echo $tipo | tr '[:upper:]' '[:lower:]' | sed 's/ //g')
173        if [[ $opt2 == "membera" ]] ; then
174           MEMBRE="a_member"
175        elif [[ $opt2 == "memberb" ]] ; then
176           MEMBRE="b_member"
177           echo $MEMBRE
178        fi
179     fi
180
181     return 0
182  }
183
184  #Création du fichier data.yaml pour remplir le template
185  creation_data_yaml () {
186     if [ $no_agent != 0 ] ; then
187       echo "org: "$org"" >> ./data.yaml
188     fi
189     echo "desc: "$descr"" >> ./data.yaml
190     echo "add: "$addr"" >> ./data.yaml
191     return 0
192  }
193
194  #Création du nouveau fichier JSON
195  creation_json () {
196     rm -f $path_SRV/mesh-central.json
197     /usr/lib/perfsonar/bin/build_json -o $path_SRV/mesh-central.json "$DIR/
           meshconfig.conf"
```

```
198     if [ $? != "0" ] ; then
199        return 1
200     fi
201     echo -e "\nLe fichier JSON a été bien créé."
202     return 0
203  }
204
205  #Sauvegarde du fichier de conf actuel pour le restaurer en cas d'echec
206  backup_fichiers () {
207     mv "$DIR/meshconfig.conf" $DIR/backup/$nomBack
208  }
209
210  #Essayez de revenir à l'état anterieur au lancement du script
211  recuperation () {
212     rm -f "$DIR/meshconfig.conf"
213     cp "$DIR/backup/$nomBack" "$DIR/meshconfig.conf"
214     if creation_json ; then
215        echo "Recupération de la config precédente réussite."
216        return 0
217     else
218        return 1
219     fi
220  }
221
222  #Rédemarrer les services perfSONAR nécessaires pour prendre en compte la
       nouvelle configuration
223  redemarrer_serv_perfsonar () {
224     echo "Redemarrage des services..."
225     systemctl restart perfsonar-meshconfig-agent
226     if [ $? != "0" ] ; then
227        return 1
228     fi
229     systemctl restart perfsonar-meshconfig-guiagent
230     if [ $? != "0" ] ; then
231        return 1
232     fi
233     systemctl restart maddash-server
234     if [ $? != "0" ] ; then
235        return 1
236     fi
```

```
237     return 0
238 }
239
240 #Appel aux scripts en Python qui feront toutes les modifications dans les
        fichiers correspondants
241 tache_add () {
242     echo -e "\nAppel au script de modification du fichier mesh config : $DIR ..."
243
244     #Cette partie permet  d'envoyer les paramètres selon le type de groupe, le
            groupe disjoint nécessite d'un paramètre en plus
245     if [[ $TYPE == "disjoint" ]] ; then
246        ./sonde_conf.py "${DIR}" "${addr}" "${TYPE}" "${no_agent}" "${MEMBRE}"
247     else
248        ./sonde_conf.py "${DIR}" "${addr}" "${TYPE}" "${no_agent}"
249     fi
250     rm -f ./data.yaml
251     backup_fichiers
252     ./creation_meshconfig.py "${DIR}"
253     return 0
254 }
255
256 tache_list () {
257     echo "" ; i=1
258     #Formatage de l'entête
259     printf "\t\t%-25s\t\t%s\n\n" "${bold}HOST" "DESCRIPTION"
260     for file in $(find $DIR/sites -type f -name *.cfg) ; do
261       nom=${file##*/}
262       if [ -z $(echo $file | grep no_agent) ] ; then
263          descr=$(grep description $file | sed -e 's/^[ ]*description//')
264       else
265          descr=$(grep -E "^ {5,}description" $file | sed -e 's/^[ ]*description
               //')
266       fi
267       printf "${normal}\t[ $i ]\t%-25s %s\n" "${nom%.*}" "$descr"               #
              Formater chaque ligne
268       (( i++ ))
269     done
270     echo ""
271 }
272
```

```
273 tache_sup_sonde () {
274   i=0
275   #Ce bucle sert à créer un tableu de facon qu'il puisse etre traité par
          whiptail
276   #Pour ca on a besoin de stocker en premier lieu le nom du fichier, ensuite son
           description et enfin l'etat de radiolist
277
278   for file in $(find $DIR/sites -type f -name *.cfg) ; do
279     fich=${file##*/}
280     sondes[i]=${fich%.*} ; (( i++ ))
281     sondes[i]="" ; (( i++ ))
282     sondes[i]="OFF" ; (( i++ ))
283   done
284
285   #On recupere la sonde choisie dans le radiolist
286   sonde_sup=$(whiptail --title "Supprimer une sonde " --radiolist "Choisissez la
           sonde que vous voulez supprimer :" 25 78 16 "${sondes[@]}" 3>&1 1>&2
          2>&3)
287
288   #Si l'utilisateur n'a rien chosi, on sort du script
289   if [[ $sonde_sup == "" ]] ; then echo "Rien" ; return 1 ; fi
290
291   #Radiolist utilise le tableau crée précedement
292   if (whiptail --title "Supprimer une sonde" --yesno "Vous êtes en train de
          supprimer la sonde $sonde_sup. Vous devez vous assurer que vous n'en avez
          plus besoin. Voulez-vous continuer ?" 8 78) then
293     lienS=$(find $DIR/groupes/ -name $sonde_sup)
294     site=$(find $DIR/sites/ -name $sonde_sup.cfg)
295     echo "Lien symbolique à supp : $lienS"
296     echo "Site a supprimer : $DIR/sites/$sonde_sup.cfg"
297     rm -f $lienS ; rm -f $site                              #Si l'utilisateur
            confirme la suppression on ecrase le lien symbolique et le site
298     echo "Mise à jour de la nouvelle configuration ..."
299     backup_fichiers
            #On sauvegarde le fichier meshconfig actuel
300     ./creation_meshconfig.py "${DIR}"
            #On crée un nouveau meshconfig
301     echo "La sonde a été bien supprimée."
302   else
303     echo "La sonde n'a pas été supprimée."
```

```
304       exit 1
305     fi
306
307     return 0
308   }
309
310   lister_param () {
311     declare -a listeArg=("${!1}")
312
313     for file in $(ls -p $DIR/tests/ | grep -v /) ; do
314       tests[i]=$(echo ${file%.*}) ; (( i++ ))
315       tests[i]="" ; (( i++ ))
316       tests[i]="OFF" ; (( i++ ))
317     done
318   }
319
320   tache_avancee () {
321       repAc=$(pwd)
322       whiptail --title "Manage tests" --yesno --yes-button "Continuer" --no-button
            "Annuler" "Dans cette section, vous allez modifier les paramétres de
           chaque test. Assurez-vous que les nouvelles valeurs sont valides." 8 78
323       if [ $? = 1 ] ; then die "Tache interrompue." 1 ; fi
324       lister_tests
325       select=$(whiptail --title "Tests trouvés" --radiolist --separate-output "\
           nChoisissez le test à modifier :" 25 78 16 "${tests[@]}" 3>&1 1>&2 2>&3)
326       if [ $? = 1 ] || [[ $select == "" ]] ; then die "Tache interrompue." 1 ; fi
327       pathSelect="$DIR/tests/$select.cfg"
328       x=0
329       for i in $(grep -E -v ">$" $pathSelect | sed -e 's/^[ ]*//' | cut -d" " -f1)
            ; do
330          param[x]=$i ;  (( x++ ))
331       done
332
333       i=0
334       for par in "${param[@]}" ; do
335         val=$(grep -E "$par" $pathSelect | sed -e 's/^[ ]*//' | cut -d" " -f2)
336         params[i]=$par ; (( i++ ))
337         params[i]="= $val" ; (( i++ ))
338         params[i]="OFF" ; (( i++ ))
339       done
```

```
340     ps=$(whiptail --title "Liste de paramètres" --checklist --separate-output "\
            nChoisissez les paramètres à modifier :" 25 78 16 "${params[@]}" 3>&1
            1>&2 2>&3)
341     if [ $? = 1 ] || [[ $ps == "" ]] ; then die "Tache interrompue." 1 ; fi
342     psAr=(${ps//" "/ })
343
344     cd $DIR/tests/
345     for i in "${psAr[@]}" ; do
346       valAc=$(grep -E "$i" $select.cfg | sed -e 's/^[ ]*//' | cut -d" " -f2)
347       val=$(whiptail --inputbox "\nEntrez le nouveau valeur pour '$i' :" 8 78
              $valAc --title "Modifier paramètre du test [$select]" 3>&1 1>&2 2>&3)
348       if [ $? = 1 ] ; then die "Tache interrompue." 1 ; fi
349       if [ $(echo $val | grep /) ] ; then
350           division=(${val//"/"/ })
351           val="${division[0]}\/${division[1]}"
352           division=(${valAc//"/"/ })
353           valAc="${division[0]}\/${division[1]}"
354       fi
355       sed -i "/$i $valAc/ c \   $i $val" "$select.cfg"
356     done
357     cd $repAc
358 }
359
360 apercu () {
361   echo  -e "Maintenant, un apercu du nouveau fichier meshconfig. "
362   read -n 1 -s -r -p "Appuyez sur une touche pour continuer : "
363   less $DIR/meshconfig.conf
364 }
365
366 #Valider les arguments passés en paramètre
367 while [ "$1" != "" ]; do
368     PARAM=$(echo $1 | awk -F= '{print $1}')
369     VALUE=$(echo $1 | awk -F= '{print $2}')
370
371     case $PARAM in
372         -h | --help)
373             aide
374             exit
375             ;;
376         --action)
```

```
377                 ACTION=$VALUE
378                 ;;
379             --dir)
380                 DIR=$(echo "$VALUE" | sed -e 's/\/$//')
381                 ;;
382             *)
383                 echo "Parametre inconnu."
384                 aide
385                 exit 1
386                 ;;
387         esac
388         shift
389     done
390
391
392     #Lancement de chaque étape en vérifiant les erreurs qui peuvent apparaître
393     echo -e "\n\nSCRIPT POUR L'ADMINISTRATION DES SONDES perfSONAR À L'OBSERVATOIRE
            ...\n"
394
395     if ! assurer_root ; then
396        die "Vous devez être superutilisateur pour exécuter ce script" 1
397     fi
398
399     if ! assurer_entres ; then
400        aide
401        die "Manque des paramètres pour le script" 1
402     else
403        dependences_script
404     fi
405
406     if ! fichiers_script_presents ; then
407        die "Manque de fichiers nécessaires pour le script. Veuillez vérifier qu'ils
               sont dans le répertoire correspondant." 5
408     fi
409
410     if [ $ACTION == "list" ] ; then
411         tache_list
412     elif [ $ACTION == "add" ] ; then
413         echo "TACHE AJOUTER UN SONDE"
414         if ! information ; then
```

```
415         die "L'installation a été interrompue." 1
416     fi
417
418     if ! creation_data_yaml ; then
419         die "Erreur produite peut-être à cause d'un probleme de droits sur le ré
                pertoire courant." 1
420     fi
421     tache_add
422     if ! creation_json ; then
423         if ! recuperation ; then
424             die "Une erreur s'est produite pendant la récuperation de la
                    configuration précedente. Vous avez le fichier $DIR.bak comme
                    backup. Là dedans, vous avez toute votre configuration MESH précé
                    dente à la MàJ esssayée." 1
425         fi
426         die "Erreur dans la création de la nouvelle configuration pour le fichier
                 JSON." 1
427     fi
428     apercu
429     if ! redemarrer_serv_perfsonar ; then
430         die "Un erreur s'est produite pendant le rédemarrage des services
                perfSONAR." 1
431     fi
432 elif [ $ACTION == "delete" ] ; then
433     echo "TACHE SUPRIMER UN SONDE"
434     if ! tache_sup_sonde ; then die "Vous devez choisir la sonde à supprimer" 1
            ; fi
435     apercu
436     if ! creation_json ; then
437         if ! recuperation ; then
438             die "Une erreur s'est produite pendant la récuperation de la
                    configuration précedente. Vous avez le fichier $DIR.bak comme
                    backup. Là dedans, vous avez toute votre configuration MESH précé
                    dente à la MàJ esssayée." 1
439         fi
440         die "Erreur dans la création de la nouvelle configuration pour le fichier
                 JSON." 1
441     fi
442     if ! redemarrer_serv_perfsonar ; then
```

```
443         die "Un erreur s'est produite pendant le rédemarrage des services
                perfSONAR." 1
444       fi
445
446   elif [ $ACTION == "conftest" ] ; then
447     if (whiptail --title "Manage tests" --yesno --no-button "Avancé" --yes-button
          "Suivant" "Normalement, cette partie est déjà configurée. Appuyez sur
          SUIVANT si vous voulez activer ou desactiver des tests ou sur AVANCÉE pour
           modifier les paramêtres des tests." 10 78) then
448       active_tests
449       ./creation_meshconfig.py "${DIR}" "${tests_mesh}" "${tests_disj}" "1"
450       apercu
451     else
452       echo "Paramètres avancés"
453       tache_avancee
454       ./creation_meshconfig.py "${DIR}" "${tests_mesh}" "${tests_disj}"
455       apercu
456     fi
457     if ! creation_json ; then
458        if ! recuperation ; then
459           die "Une erreur s'est produite pendant la récuperation de la
                  configuration précedente. Vous avez le fichier $DIR.bak comme backup
                  . Là dedans, vous avez toute votre configuration MESH precédente à
                  la MàJ esssayée." 1
460        fi
461        die "Erreur dans la création de la nouvelle configuration pour le fichier
               JSON." 1
462     fi
463     if ! redemarrer_serv_perfsonar ; then
464        die "Un erreur s'est produite pendant le rédemarrage des services perfSONAR
               ." 1
465     fi
466   elif [ $ACTION == "meshfile" ] ; then
467       apercu
468   else
469       aide
470   fi
471   exit 0
```

6.3 Script creation_meshconfig.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Ce script permet de créer la configuration finale meshconfig.conf, avec tous les blocs nécessaires (organisations, sites, tests, groupes, etc). Il va parcourir toute l'arborescence du MESH pour retrouver les données requises

import os
from jinja2 import Environment, FileSystemLoader
import yaml
import sys
import glob

testsMeshCourants = {}
testsDisjCourants = {}
testsMeshNew = []
testsDisjNew = []

#Cette function va creer tous les tests en fonction des paramètres passés
#liste = array  contenant les test passes pour le script bash
#courants = array  contenant les test tournant à présent
#groupe = le groupe relie au test
#title = INTERNE ou EXTERIEUR cest juste pour l'affichage
#modif = 1 ou 0, 1 si on a lance directement l'action 'conftest'
#0 si on vient des taches 'add' ou 'delete' sonde

def test (liste,courants,groupe,title,modif):
    for test in liste:
        file.write("<test>\n")
        if modif == 1:
            if test in courants:
                 descrip = courants[test]
            else:
                descrip = raw_input("GROUPE " + title + " : Entrez une description pour le nouveau test : " + test + " - ")
        else:
            descrip = courants[test]
        file.write("   description    " + descrip + "\n")
```

```
36          file.write("   group    " + groupe + "\n")
37          file.write("   test_spec    " + test + "\n")
38          file.write("</test>\n\n")
39          cmd = 'echo "' + groupe + ',' + test + ',' + descrip + '" >> ' +
                reperTests
40          os.system(cmd)
41
42  def lister_tests_courants ():
43      #On lit le fichier actives.cfg pour recuperer les tests selectionés
44      for line in open(reper + "/tests/actives/actives.cfg").readlines():
45          lin = line.rstrip('\n').split(',')
46          if lin[0] == "obas_interne_mesh":
47              #testsMeshCourants[lin[1]] = lin[2]
48              testsMeshCourants[lin[1]] = lin[2]
49          else:
50              testsDisjCourants[lin[1]] = lin[2]
51
52  #On récupère le chemin du répertoire MESH grace au script BASH
53  reper = sys.argv[1]
54  nomFich = reper + "/meshconfig.conf"
55
56  if len(sys.argv) == 5:
57      TESTS1 = sys.argv[2]
58      TESTS2 = sys.argv[3]
59      testsMeshNew = TESTS1.split('\n')
60      testsDisjNew = TESTS2.split('\n')
61      modif = 1
62  else:
63      modif = 0
64
65  lister_tests_courants()
66
67  entete = """description PerfSONAR Observatoire Mesh
68
69  <organization>
70      description     Observatoire Astronomique de Strasbourg
71
72  """
73
74  #On crée le nouveau fichier de configuration dans le chemin spécifié
```

```
75  file = open(nomFich,"wb")
76
77  file.write(entete)
78
79  for fich in glob.glob(reper + '/sites/*.cfg'):
80      #On lit chaque fichier de conf et l'écrit dans le fichier meshconfig
81      for line in open(fich).readlines():
82          file.write(line)
83
84  #On ferme le bloc organisation
85  file.write("</organization>\n\n")
86
87  for fich in glob.glob(reper + '/sites/no_agent/*.cfg'):
88      #On lit chaque fichier de conf et l'écrit dans le fichier meshconfig.conf
89      for line in open(fich).readlines():
90          file.write(line)
91      file.write("\n\n")
92
93  for fich in glob.glob(reper + '/tests/*.cfg'):
94      #On lit chaque fichier de conf et l'écrit dans le fichier meshconfig
95      for line in open(fich).readlines():
96          file.write(line)
97      file.write("\n")
98
99  #On commence la partie des groupes
100
101 #On utilise la même méthode pour obtenir les informations : dans ce cas, on
        parcours l'arborescence en cherchant les les membres du groupe mesh et les é
        crit dans le fichier
102 file.write("<group obas_interne_mesh>\n")
103 file.write("   type mesh\n")
104 for mem in glob.glob(reper + "/groupes/mesh/*"):
105     dirname, filename = os.path.split(mem)
106     file.write("\n   member " + filename)
107 file.write("\n</group>")
108
109 file.write("\n\n<group obas_exterieur_disj>\n")
110 file.write("   type disjoint\n\n")
111 types = [os.path.basename(x) for x in glob.glob(reper + "/groupes/disjoint/*")]
112 types.sort()
```

```
113 for tipo in types:
114     for mem in glob.glob(reper + "/groupes/disjoint/" + tipo + "/*"):
115         dirname, filename = os.path.split(mem)
116         file.write("   " + tipo + " " + filename + "\n")
117     file.write("\n")
118 file.write("</group>\n\n")
119 
120 reperTests = reper + '/tests/actives/actives.cfg'
121 cmd = 'rm -f ' + reperTests
122 os.system(cmd)
123 
124 #Appel à la fonction de création des blocs de test
125 if modif == 1:
126     print "\nMaintenant, vous devez entrer les descriptions pour chacun des
        nouveaux tests. Ces descriptions sont celles affichées dans le tableau
        de bord, il faut donc donner des descriptions adaptées.\n"
127     test(testsMeshNew,testsMeshCourants,"obas_interne_mesh","INTERNE",1)
128     test(testsDisjNew,testsDisjCourants,"obas_exterieur_disj","EXTERIEUR",1)
129 else:
130     testsMesh = testsMeshCourants.keys()
131     testsDisj = testsDisjCourants.keys()
132     test(testsMesh,testsMeshCourants,"obas_interne_mesh","INTERNE",0)
133     test(testsDisj,testsDisjCourants,"obas_exterieur_disj","EXTERIEUR",0)
134 
135 print "\nConfiguration complète !\n"
136 
137 file.close
```

6.4 Script sonde_conf.py

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 
4 #Ce script permet de créer la configuration de chaque nouvelle sonde et la
    placer dans le répertoire correspondant. De plus, il crée un lien symbolique
     vers le répertoire groupes pour les 'relier'
5 
6 import os
7 from jinja2 import Environment, FileSystemLoader
```

```
import yaml
import sys

rep = sys.argv[1]
idSonde = sys.argv[2]
grp = sys.argv[3]
no_agent = sys.argv[4]
grpPath = "/"

print "no_agent = ", no_agent
#Si on recoit cinq paramètres ca veut dire qu'il s'agit d'un groupe disjoint
if len(sys.argv) == 6:
    membre = sys.argv[5]
    #On crée le chemin vers tous les fichiers du groupe disjoint
    grpPath = "/" + membre + "/"

if no_agent == '0':
    #On construit le nom complet du fichier de la nouvelle configuration du site
    nomFich = rep + "/sites/" + idSonde + ".cfg"
else:
    nomFich = rep + "/sites/no_agent/" + idSonde + ".cfg"

dest = rep + "/groupes/" + grp + grpPath + idSonde

#On crée la commande pour le lien symbolique
cmd = "ln -s " + nomFich + " " + dest

#On obtient les données du fichier YAML
config_data = yaml.load(open('./data.yaml'))

#On utilise le template pour créer la configuration
env = Environment(loader = FileSystemLoader('./'), trim_blocks=True,
    lstrip_blocks=True)
if no_agent == 0:
    template = env.get_template('sonde_obas.temp')
else:
    template = env.get_template('no_agent.temp')
config = template.render(config_data)

#On l'écrit dans le fichier
```

```
file = open(nomFich,"wr")
file.write(config)

#On crée le lien simbolique
os.system(cmd)
```

6.5 Template sonde_obas.temp

```
    <site>
        <host>
              description {{ desc }}
              address {{ add }}

              <measurement_archive>
                    type traceroute
                    read_url http://{{ add }}/esmond/perfsonar/archive
                    write_url http://{{ add }}/esmond/perfsonar/archive
              </measurement_archive>

              <measurement_archive>
                    type perfsonarbuoy/owamp
                    read_url http://{{ add }}/esmond/perfsonar/archive
                    write_url http://{{ add }}/esmond/perfsonar/archive
              </measurement_archive>

              <measurement_archive>
                    type perfsonarbuoy/bwctl
                    read_url http://{{ add }}/esmond/perfsonar/archive
                    write_url http://{{ add }}/esmond/perfsonar/archive
              </measurement_archive>

              <measurement_archive>
                    type pinger
                    read_url http://{{ add }}/esmond/perfsonar/archive
                    write_url http://{{ add }}/esmond/perfsonar/archive
              </measurement_archive>
        </host>
    </site>
```

6.6 Template no_agent.temp

```
<organization>
    description  {{ org }}

    <site>
        <host>
              no_agent 1
              description {{ desc }}
              address {{ add }}
        </host>
    </site>
</organization>
```