Laboratoire d'informatique de Grenoble

NGUYEN Justin
INFORMATIQUE POLYTECH GRENOBLE
Rapport de stage 2021/2022.

Faults in Linux

Tome Principal
ET
Annexes

Année universitaire : 2021/2022
Période du stage : du 9 mai au 26 août 2022

Remerciements

Je tiens à remercier le Laboratoire d'Informatique de Grenoble pour l'accueil, en particulier l'équipe ERODS avec qui j'ai pu travailler.

Je remercie également Nicolas PALIX, mon tuteur de stage, qui m'a encadré et conseillé pendant l'intégralité du stage, et pour son aide à la structuration de ce rapport.

Pour finir, je remercie Jean-François MONIN en tant qu'enseignant référent pour m'avoir suivi durant la totalité de mon stage.

Table des Matières

Index des figures

Abstracts

Version française

Pour mon stage d'INFO4 j'ai réalisé un stage en recherche au Laboratoire d'Informatique de Grenoble proposé par l'enseignant Nicolas PALIX qui fut également mon tuteur de stage. Je faisait partie de l'équipe ERODS. Le sujet de stage portait sur l'étude *Fautes dans Linux* qui concernait l'emplacement et la fréquence des fautes dans le noyau Linux. Pour cela, il fallait étendre aux versions 3.X, 4.X et 5.X du noyau Linux l'étude et également mettre à jour l'outil Herodotos. Pour mettre à jour Herodotos, j'ai tout d'abord passé sous un nouvel environnement, Dune, qui permet de gérer les librairies OCaml bien plus facilement. Ensuite pour étendre aux versions ultérieures de Linux l'étude, j'ai réalisé des modifications sur les makefiles de récupération de version et de comptage de nombre de lignes de code source. J'ai également mis à jour la documentation pour Herodotos en LaTeX pour que le travail de mise à jour puisse être effectué par d'autres collaborateurs du sujet grâce à un listage des dépendances avec une machine virtuelle. De l'intégration continue a également été effectuée avec plusieurs dépôts git et un déploiement sur les machines distantes de Grid'5000. J'ai également travaillé avec un autre stagiaire venant de l'Ensimag sur le même sujet. Le travail qui nous était individuellement donné était assez différent pour que je ne puisse pas décrire en détail ce que mon binôme a fait durant le stage.

English version

For my internship in INFO4 I did a research assignment at the Grenoble Computer Science Laboratory proposed by the teacher Nicolas PALIX who was also my internship tutor. I was part of the ERODS team. The subject of the internship was the study of faults in Linux which concerned the location and frequency of faults in the Linux kernel. For that, it was necessary to extend the study to the versions 3.X, 4.X and 5.X of the Linux kernel and also to update the Herodotos tool. To update Herodotos, I first switched to a new environment, Dune, which allows to manage OCaml libraries much more easily. Then to extend the study to later versions of Linux, I made modifications to the makefiles for version retrieval and for counting the number of source lines of code. I also updated the documentation for Herodotos in LaTeX so that the maintenance work could be done by other contributors to the topic through a dependency listing with a virtual machine. Continuous integration was also done with several git repositories and deployment on remote Grid'5000 machines. I also worked with another intern from Ensimag on the same subject. The work we were individually given was different enough that I cannot describe in detail what my partner did during the internship.

Introduction

Linux est un système d'exploitation utilisé par de nombreux systèmes, allant de systèmes embarqués soumis à de nombreuses contraintes de portabilité à des superordinateurs nécessitant une puissance de calcul impressionnante, en passant par des machines personnelles et des serveurs, l'utilisation de Linux est globale et généralisée.

Le noyau Linux est un noyau de système d'exploitation de type UNIX et est responsable de la sécurité et l'intégrité des interactions logicielles et matérielles sur le système d'exploitation. C'est donc pour cela qu'il est essentiel de s'assurer de sa fiabilité.

Le Laboratoire d'Informatique de Grenoble (LIG) fut crée en 2007. C'est un laboratoire de recherche français en informatique et il est sous la tutelle à la fois de l'Université Grenoble Alpes, l'institut polytechnique de Grenoble et du CNRS. Il est également en partenariat avec l'INRIA.

Il possède de nombreuses équipes travaillant chacune dans différents domaines de l'informatique,

notamment l'équipe ERODS (Efficient and RObust Distributed Systems) dont mon tuteur fait partie. L'objectif de l'équipe étant d'étudier la construction et l'administration d'infrastructures de Cloud Computing ayant pour thèmes scientifiques principaux : systèmes autonomes, support distribués et les machines virtuelles.

Mon stage dure du 9 mai au 26 août, avec 7 heures de travail par jour (souvent de 9h à 17h), 5 jours par semaines avec une interruption du 1 août au 15 août car le laboratoire ferme à cette période.

J'étais placé dans le bureau 474 pendant le premier mois de stage avec d'autres stagiaires puis on m'a déplacé au bureau 269 car il n'y avait plus de place dans le bureau. J'ai finalement fini en Juillet au bureau 469 jusqu'à la fin de mon stage.

J'ai travaillé avec un autre stagiaire venant de l'Ensimag sur le même sujet à partir de la seconde semaine de stage.

Pour la structure de ce rapport :

Je commencerai tout d'abord par présenter mon sujet de stage et les deux principaux outils utilisés.

Ensuite j'enchaînerai avec le travail effectué pendant le stage.

Je finirai par un bilan au moment de la rédaction de ce rapport et un bilan personnel suivi d'une rapide conclusion.

I - Fautes dans Linux

Présentation du sujet

En 2001, une première étude a été publiée par Chou *et al.* portant sur les fautes trouvées par analyse statique dans les versions 1.0 à 2.4.1 de Linux, ce qui a permis de révéler que le répertoire drivers contenait sept fois plus de fautes que les autres répertoires étudiés. Malgré les travaux de recherches découlant de l'étude qui ont permis d'améliorer la stabilité du noyau, Linux évolue d'autant plus vite avec des fonctionnalités, des utilisations bien plus variés et un tout nouveau modèle de développement.

C'est pour cela qu'en 2011, l'étude *Fautes dans Linux, dix ans après* a vu le jour. En effet les nombreux changements ont généré d'autant plus d'interrogations. Il fallait vérifier l'impact de ces différences sur la qualité du code et si les pilotes étaient toujours le dossier le plus problématique. Ainsi l'étude de 2001 a été étendue aux versions de Linux 2.6.0 à 2.6.33, versions parues de 2003 à 2010.

Les observations retenues par l'étude étaient que la taille du code avait plus que doublé mais que le nombre de fautes par ligne de code a diminué , le dossier des pilotes contenait toujours le plus de fautes mais que sont taux de fautes était inférieur à celui d'autres répertoires.

Le but du stage était premièrement d'évaluer la reproductibilité de l'étude de 2011 et d'identifier les éventuels problèmes et deuxièmement d'évaluer la présence d'une tendance dans le type de fautes et leur emplacement tout en vérifiant l'impact de l'intégration et des tests continus sur les résultats.

Les outils majeurs utilisés dans l'étude sont coccinelle et Herodotos.

Dans cette optique, il faudra d'abord mettre à jour Herodotos, ce qui nécessite de comprendre son fonctionnement pour le rendre de nouveau fonctionnel. Ensuite, il faudra créer une image contenant les outils et la déployer sur Grid'5000 pour utiliser les machines mises à disposition, optimiser les traitements en parallélisant certaines phases et usant au maximum les multiples nœuds dans grid'5000.

Le dépôt faults-in-linux centralise l'utilisation des outils.

Présentation de l'outil coccinelle

Coccinelle est un programme de correspondance et de transformation (matching and transformation engine) et mets à disposition le langage SmPL (Semantic Patch Language) afin de spécifier les correspondances et les transformations désirées en C.
Initialement, coccinelle fut pensé pour réaliser l'évolution collatérale de Linux pour faciliter l'évolution du code des pilotes qui sont soumis à la progression des APIs des librairies telles que le renommage de fonctions, l'ajout d'arguments de fonctions dont la valeur dépends du contexte et l'arrangement d'une structure de données.
En plus de l'évolution collatérale, coccinelle est utilisé pour rechercher et corriger des bogues dans le code système.

Coccinelle s'utilise en appliquant des patchs sémantiques, qui, à la différence des patchs traditionnels, peuvent modifier des centaines de fichiers et des milliers de zones de code.

Présentation de l'outil Herodotos

Herodotos est un outil pour la recherche de motifs de code parmi différentes versions d'un programme et d'en établir l'historique. Il s'utilise avec des lignes de commande.

Herodotos réalise plusieurs phases lors de son utilisation :

1. preinit : Herodotos génère un fichier de cache des versions du programme spécifié dans le fichier .hc (herodotos configuration)
2. init : Herodotos génère les dossiers de résultats ainsi que des fichiers .orig.org et des dépendances makefile
3. correl : Herodotos effectue la corrélation en modifiant les fichiers org
4. graph : Herodotos génère les graphes correspondants aux résultats
5. stat : calcule les statistiques générales
6. statcorrel : calcule les statistiques des corrélations
7. statfp : calcule les statistiques des faux positifs

Et bien d'autres phases qui sont en réalité des options à préciser lors de l'entrée des commandes dans le terminal.

L'outil est composé de plusieurs sous-outils/dépendances :

1. Coccinelle : L'outil coccinelle permet d'appliquer des patchs sémantiques spécifiques pour réaliser la phase de corrélation de Herodotos
2. hBugs : permet de calculer les paramètres thêta(inconnues)
3. Menhir : générateur de parseur pour le langage OCaml
4. Volt : outil de logging pour OCaml
5. Parmap : librairie minimaliste pour exploiter les architectures multicœurs pour les programmes OCaml

II – Travail effectué

Mise à jour de Herodotos

Afin de mettre à jour Herodotos, j'ai tout d'abord commencé par réaliser des portages simples :

1. hBugs : Pour réaliser une mise à jour simple, j'ai lancé la compilation à l'aide de make et j'ai regardé les erreurs qui s'affichaient sur le terminal, je me suis ensuite renseigné sur le wiki Haskell pour mettre à jour le code (un changement de syntaxe des Monads)

2. documentation LaTeX : j'ai importé le dossier docs sur Overleaf pour l'éditer, en cas de doutes pour la syntaxe, j'ai pu vérifier sur internet donc je n'ai pas eu de problèmes.

3. Scripts de configuration/installation : afin de savoir quelles seraient les nouvelles dépendances et celles qui étaient devenues obsolètes, j'ai installé sur une machine virtuelle Ubuntu 20.04.4 LTS et j'ai vérifié la version de chaque paquet pour faire fonctionner le projet. La modification des scripts était assez simple car il suffisait de mettre les commandes que j'ai entrées lors de mes tests sur la machine virtuelle.

Ensuite, pour faire fonctionner Herodotos, j'ai suivi les indications du readme inclus dans le code source mais menhir avait besoin que du flag *--infer* pour inférer les types dans la génération de

parseurs, cependant même avec cela , menhir ne reconnaissait pas les modules OCaml présents dans le même dossier, ce qui était assez problématique.

C'est ainsi que j'ai décidé de porter sur l'environnement dune Herodotos, car c'était une des propositions du message d'erreur de Menhir.

Pour ce faire, il faut installer dune grâce au gestionnaire de paquets opam.

Ensuite il faut initialiser le projet dune même si il y a des fichiers déjà présents avec la commande *dune init proj <nom_du_projet>*

Cela va créer un nouveau dossier nommé *nom_du_projet* avec à l'intérieur des sous dossiers et des fichiers dune pour les librairies.
Les fichiers dune étant :

1. dune : fichier qui définit le dossier dans lequel il se trouve, avec un stanza executable, la compilation utilisera le contenu du dossier pour compiler un exécutable, avec un stanza library, le dossier sera utilisé comme une librairie qui sera référencée dans d'autres fichiers dunes en tant que librairie.
2. Dune-project : fichier qui donne diverses informations pour générer le fichier opam du projet, telles que la version de dune , de menhir, les informations du paquets, etc.

Cependant, même avec dune, le projet ne voulait pas terminer sa compilation car les modules OCaml n'étaient pas reconnus entre eux lorsqu'ils étaient dans des dossiers différents.
C'est pour cela qu'au départ j'avais mis à plat tous les dossiers du projet, mais ce n'était pas la meilleure des idées car le projet perdait de sa modularité et c'était difficile de naviguer parmi tous les fichiers.
J'ai ensuite gardé l'arborescence en sous-dossiers initiale et ajouté des fichiers dune pour chacun des dossiers pour en faire des librairies mais certains fichiers avaient besoin de fichiers d'un autre dossiers mais les fichiers de ce dossier dépendaient d'eux aussi ce qui provoquait une dépendance cyclique qui n'avait pas lieu d'être.
Il existe une solution pour ne pas avoir besoin de créer des fichiers dune pour les librairies : include_subdirs ce qui règlerait les problèmes de dépendance cyclique et éviterait la création d'une dizaine de fichiers dune en plus mais malheureusement, menhir n'est pas compatible avec cette directive ce qui a été identifié en tant que bug non résolu à l'écriture de ce rapport. Alors pour régler ce problème, j'ai fusionné les dossiers problématiques même si on perds la modularité de deux dossiers pour que ça fonctionne enfin.
Cependant il y avait encore des problèmes: les lignes pour l'utilisation de l'outil de logging volt n'étaient pas reconnues en tant que tel. J'ai donc changé chacune des lignes en syntaxe implicite qui nécessitaient du preprocessing en syntaxe explicite qui ne nécessitait plus de preprocessing.

```
"--profile", Arg.Unit (fun () -> Bolt.Logger.log "" Bolt.Level.TRACE "*** PROFILING ENABLED ***" ;
                        Debug.profile := Debug.PALL), " gather timing information about the main functions";
```

```
"--profile", Arg.Unit (fun () -> LOG "*** PROFILING ENABLED ***" LEVEL TRACE;
                        Debug.profile := Debug.PALL), " gather timing information about the main functions";
```

Figure 1: Comparaison des syntaxes de Bolt/Volt

Mais si j'avais choisi de garder la syntaxe implicite, j'aurais du spécifier une directive dans chacun des fichiers dune de librairies et d'exécutable quels étaient les fichiers nécessitant du preprocessing.
Et même si cela avait abouti, j'aurais rencontré une erreur de constructeurs currifiés qui nécessiterait de changer le code de Herodotos et de se débarrasser de tous les constructeurs, ce qui me paraissait inenvisageable et contre-productif.
C'est pour cela que je considère ma solution comme acceptable

Pour tester si Herodotos fonctionnait, j'ai édité le fichier test.hc avec différentes versions de linux ajoutées manuellement. J'ai pu observer que le logging ne fonctionnait pas au départ, car l'ordre des arguments entre la syntaxe implicite et explicite n'est pas la même.
En exécutant les différentes phases de Herodotos, j'ai pu observé les différents fichiers générés et modifiés.

Extension aux versions ultérieures de Linux

Le dépôt npalix/linux-infra contient le makefile pour récupérer les différentes versions de linux et les scripts pour compter le nombre de lignes de code source de chacune des versions.

Pour récupérer les versions de linux qui m'intéressaient j'ai modifié le makefile pour qu'il ne télécharge que les tags contenant les versions sélectionnées.

```
TAGS3=$(shell (git ls-remote --tags $(LINUSTREE) v3.[0-9] v3.[0-9][0-9] | sed 's/^.*\(v3.*\).*$$/\1/'))
TAGS4=$(shell (git ls-remote --tags $(LINUSTREE) v4.[0-9] v4.[0-9][0-9] | sed 's/^.*\(v4.*\).*$$/\1/'))
TAGS5=$(shell (git ls-remote --tags $(LINUSTREE) v5.[0-9] v5.[0-9][0-9] | sed 's/^.*\(v5.*\).*$$/\1/'))
#You can change the tags to download other versions :
TAGS += $(TAGS5)
all: update-git
        $(MAKE) check-all
check-all: $(DIR)

linux-git:
        git clone $(LINUSTREE) $@
        for i in $(TAGS); do \
                git clone --depth 1 --branch $$i $(LINUSTREE) $@ ; \
                git --git-dir=linux-git/.git fetch ; \
                $(MAKE) ; \
                rm -rf linux-git ; \
        done
```

Figure 2: Modifications effectuées dans le makefile

Cependant, ma solution supprime à chaque itération de boucle le dossier git (je ne garde que les versions en tant que telles) ce qui est problématique pour coccinelle/fault-in-linux donc j'ai arrêté de l'utiliser.

Pour compter le nombre de lignes de codes sources, l'outil sloccount est utilisé. Au départ il était utilisé dans une série de scripts shell, donc j'ai choisi d'adapter cela en makefile pour qu'il vérifie la présence de sloccount avant de commencer à traiter les versions.

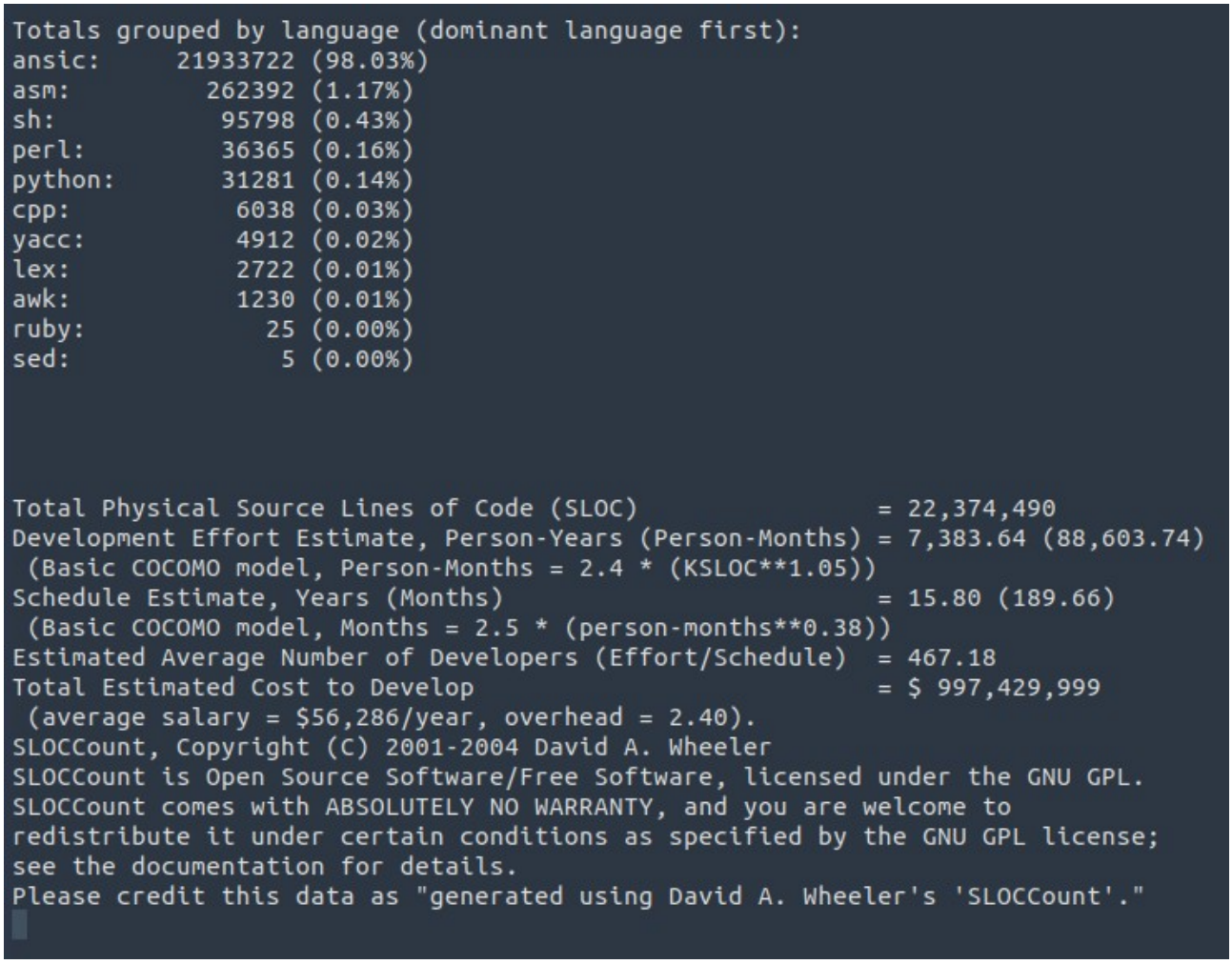

Figure 3: Sortie des scripts pour compter les lignes de code source (Linux 5.18)

Make permet de générer un fichier que s'il n'est pas encore présent ce qui est pratique pour éviter de traiter inutilement des versions déjà traitées.

Pour tester si les versions avaient bien été récupérées, je les ai utilisées dans Herodotos en modifiant le fichier de test.hc.
Pour tester si les lignes de codes étaient bien comptées j'ai juste comparé les sorties de versions dont la taille avait déjà été comptée avec les résultats précédents.

Intégration continue et tests

Afin de s'assurer que le projet puisse évoluer à chaque mise à jour, et que de multiples contributions puissent être effectuées, le projet est séparé en plusieurs dépôts git : coccinelle/faults-in-linux , coccinelle/herodotos et npalix/linux-infra.

Git permet à plusieurs personnes de collaborer sur un même projet simultanément et d'y ajouter la contribution de chacun.

J'ai crée une branche nommée tout d'abord *dune_WIP* sur le dépôt coccinelle/herodotos lors de mes premiers changements mais mes commits étaient trop regroupés et volumineux. J'ai dû créer une branche *dune_2_* afin de séparer mes commits et aussi nettoyer les fichiers générés qui ne devaient pas être présents dans le dépôt.

En ce qui concerne grid'5000, j'ai juste configuré mon tunnel ssh car je n'ai pas vraiment eu besoin d'utiliser les machines distantes.
En revanche, mon binôme a dû utiliser grid'5000 à maintes reprises afin de tester ses scripts.

Pour ce qui en est de la coopération avec mon binôme, nous communiquons dès que nous rencontrions des problèmes afin d'échanger nos idées de solutions, lors des réunions pour faire le point avec notre tuteur, des notes de toutes les tâches et suggestions étaient prises mêmes si elles nous concernaient pas individuellement, afin d'avoir le contexte de travail de chacun. De plus, avoir des précisions sur des outils spécifiques peut toujours être utile même si on n'utilise pas ces outils immédiatement.

Pour réaliser les tests, avant de tester directement sur les machines de grid'5000, on testait d'abord localement sur nos machines personnelles et comparons nos résultats. Si l'un de nous obtenait des erreurs que l'autre n'avait pas, on cherchait la source de l'erreur afin d'éviter qu'elle se produise lors du déploiement sur les machines distantes.

Le stage n'est pas encore terminé, je continuerai à explorer les possibilités d'optimisation de Herodotos grâce à la librairie Parmap.

III – Bilan du stage

État final du projet au moment de la rédaction du rapport

Outre les tâches qui m'étaient données de faire à chaque réunion hebdomadaires, j'ai travaillé en autonomie et j'étais libre de choisir la gestion du projet. J'ai pu avancer avec une planification hebdomadaire avec une liste de tâches que je voulais effectuer.

Cependant, même avec cela, je n'ai pas pu réaliser l'intégralité de ce que je voulais faire.

La mise à jour de Herodotos a été effectuée, et la rédaction de la documentation permettra à de futurs collaborateurs de garder à jour l'outil.

L'évaluation de la reproductibilité de l'étude à été partiellement établie : on peut générer les mêmes types de graphes à l'aide de Herodotos, les versions antérieures du noyau Linux son toujours disponibles sur le dépôt git, il ne reste plus qu'à continuer l'étude avec le travail effectué par mon binôme sur Grid'5000.

Cependant il reste à voir si les résultats des versions ultérieures du noyau Linux seront cohérents ou non. Et également chercher des pistes d'optimisation de Herodotos avec Parmap et une optimisation sur les machines distantes de Grid'5000.

Connaissances acquises et réflexion sur le stage

La compétence C2 de l'ingénieur INFO : Concevoir, développer et intégrer des briques logicielles fut la principale compétence développée durant ce stage.

La mise à jour de Herodotos m'a permis d'actualiser un outil déjà existant en regardant d'abord de quelle manière il s'utilisait, de lister les mécanismes utilisés et dont il dépendait. Cela m'a permis d'établir une liste des dépendances à placer dans la documentation mais aussi mettre à jour celle-ci.

De plus j'ai dû regarder quels composants étaient à adapter pour les mettre à jour : le passage à l'environnement dune était dans ce but, pour avoir une liste claire des librairies utilisées.

J’ai pu appliquer les connaissances déjà acquises durant ma formation en OCaml, git et make, mais surtout les revoir et les approfondir.

Par exemple commit sur un dépôt déjà existant, créer ma propre branche et faire des changements en fonction des commentaires reçus (revenir en arrière sur l’historique git avec un soft reset était une première pour moi).

J’ai découvert le travail de recherche en laboratoire pour la première fois.

J’ai également pu interagir avec d’autres stagiaires du laboratoire et leur demander de me parler de leur sujet de stage par curiosité.

Ce stage m’a été très bénéfique pour approfondir mes connaissances mais également revoir ce que j’ai appris durant ma formation.

Travailler sur les fautes dans Linux a toujours eu de l’intérêt à cause de l’utilisation globale du noyau Linux et également que l’apparition de fautes est lié à l’ajout de code de fonctionnalité donc le sujet est toujours pertinent.

En rétrospective de mon travail, je pense que j’aurais pu avancer plus rapidement si je maîtrisais les outils utilisés, j’aurais dû solliciter plus souvent l’aide de la part de mon tuteur même si j’étais à un étage différent du bâtiment, j’aurais dû faire un planning des tâches à effectuer pour mieux m’organiser également.

Conclusion

Ce stage fut globalement très positif de mon point de vue, j’ai beaucoup appris et j’ai quand même pu produire un travail utilisable. L’encadrement de la part de mon tuteur était également agréable avec des réunions hebdomadaires pour contrôler ma progression mais assez de liberté pour choisir mon rythme de travail. Le fait d’avoir à tenir un journal de bord m’a également beaucoup aidé durant le stage et pour la rédaction de ce rapport.

Annexes

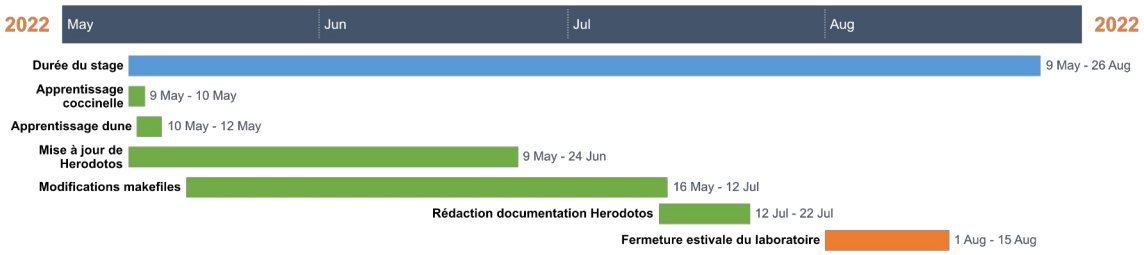

Figure 4: Diagramme de Gantt de mon stage (noter que mise à jour de Herodotos et modifications makefiles devaient contenir des sous tâches)

DOS DU RAPPORT

Etudiant (nom et prénom) : NGUYEN Justin Année d'étude dans la spécialité : 4

Entreprise : UNIVERSITE GRENOBLE ALPES - LIG - ERODS
Adresse complète : (géographique et postale)
621 AV CENTRALE 38400
SAINT-MARTIN-D'HERES
Téléphone (standard) : 04 57 42 21 42
Responsable administratif (nom et fonction) :
Mr LAKHNECH Yassine Président
Téléphone : 04 76 01 27 02
Courriel: Yassine.Lakhnech@univ-grenoble-alpes.fr

Tuteur de stage (organisme d'accueil)
Mr PALIX Nicolas
Téléphone : 04 57 42 15 38
Courriel: nicolas.palix@imag.fr

Enseignant-référent :
Mr MONIN Jean-François
Téléphone : 06 81 23 30 05
Courriel: jean-francois.monin@univ-grenoble-alpes.fr

Titre : (maximum 2 à 3 lignes).
Faults in Linux

Résumé : (minimum 15 lignes).
Pour mon stage d'INFO4 j'ai réalisé un stage en recherche au Laboratoire d'Informatique de Grenoble proposé par l'enseignant Nicolas PALIX qui fut également mon tuteur de stage. Je faisait partie de l'équipe ERODS. Le sujet de stage portait sur l'étude *Fautes dans Linux* qui concernait l'emplacement et la fréquence des fautes dans le noyau Linux. Pour cela, il fallait étendre aux versions 3.X, 4.X et 5.X du noyau Linux l'étude et également mettre à jour l'outil Herodotos. Pour mettre à jour Herodotos, j'ai tout d'abord passé sous un nouvel environnement, Dune, qui permet de gérer les librairies OCaml bien plus facilement. Ensuite pour étendre aux versions ultérieures de Linux l'étude, j'ai réalisé des modifications sur les makefiles de récupération de version et de comptage de nombre de lignes de code source. J'ai également mis à jour la documentation pour Herodotos en LaTeX pour que le travail de mise à jour puisse être effectué par d'autres collaborateurs du sujet grâce à un listage des dépendances avec une machine virtuelle. De l'intégration continue a également été effectuée avec plusieurs dépôts git et un déploiement sur les machines distantes de Grid'5000. J'ai également travaillé avec un autre stagiaire venant de l'Ensimag sur le même sujet. Le travail qui nous était individuellement donné était assez différent pour que je ne puisse pas décrire en détail ce que mon binôme a fait durant le stage.