Projet TuxML

Compilation du *kernel* Linux et *Machine Learning*

Corentin Chédotal Gwendal Didot Dorian Dumanget
Erwan Le Flem Pierre Le Luron Alexis Le Masle
Mickaël Lebreton Fahim Merzouk

Master 1 INFORMATIQUE
Université Rennes 1

10 janvier 2018

Table des matières

Introduction
Enjeux

Introduction
Enjeux

Le *kernel* Linux

- Noyau Linux utilisé dans le monde entier (serveurs, box Internet, Android...)
- Plus gros projet open-source au monde
  $\Rightarrow$ Premier projet sur GitHub
  $\Rightarrow$ Milliers de contributeurs
  $\Rightarrow > 700000$ *commits*

Introduction
Enjeux

Le *kernel* Linux

- Noyau Linux utilisé dans le monde entier (serveurs, box Internet, Android...)
- Plus gros projet open-source au monde
  $\Rightarrow$ Premier projet sur GitHub
  $\Rightarrow$ Milliers de contributeurs
  $\Rightarrow > 700000$ *commits*

Problèmes

- Nombreuses options (plus de 14000)
- Impact et interactions entre les options méconnus, y compris pour les développeurs

Introduction
Machine Learning

- Apprentissage automatique
- Approche prédictive
- Peut nécessiter de très nombreuses données (dépend de la complexité du problème)
- Cycle de 3 étapes : *sampling*, *testing*, *learning* puis on recommence avec nos nouvelles données

Le but du projet

Associer le Machine Learning à la compilation du noyau Linux.
Quelques objectifs :

- Détecter des interactions non documentées entre options
- Comprendre l'impact de certaines options sur des mesures physiques du noyau
- A terme, proposer des configurations pré-faites pour accomplir au mieux une requête de très haut niveau

$\Rightarrow$ Améliorer le noyau Linux

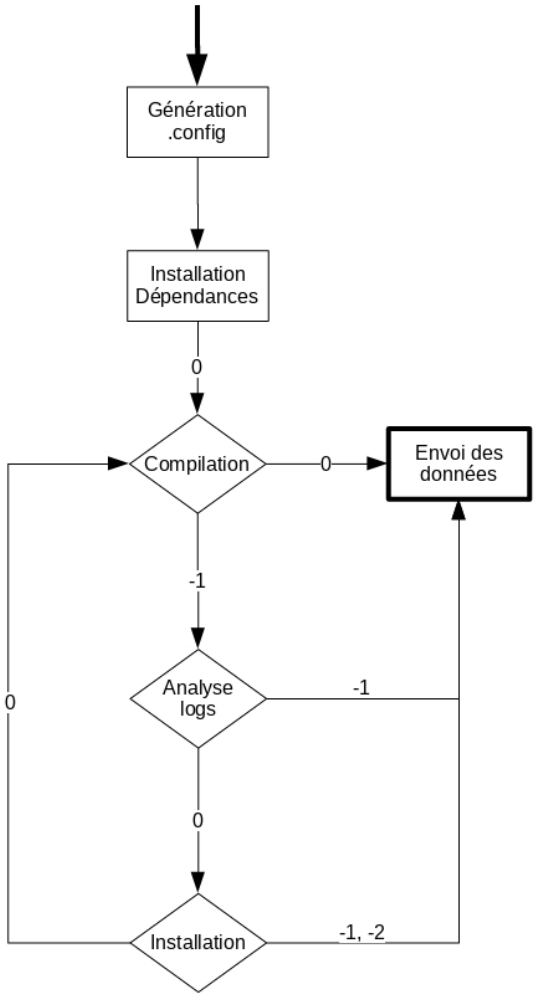
Quelques explications sur la compilation...

- Structure du `.config`
- Utilisation de `make`
- Obtention d'un fichier image

Architecture globale

- Gestion des dépendances
- Compilation automatique
- Envoi à une base de données

Scripts
tuxml.py



Scripts
tuxml.py

```
[*] 2018-01-09 19:43:36 | Cleaning previous compilation
[*] 2018-01-09 19:43:53 | Randomising new config file
[*] 2018-01-09 19:44:00 | Getting environment details
    ==> system
      --> distrib_version: 9.3
      --> os: Linux
      --> distribution: debian
      --> kernel: 4.9.0-3-amd64
    ==> compilation
      --> gcc_version: 6.3.0-18)
      --> core_used: 8
      --> libc_version: 2.24-11+deb9u1)
      --> incremental_mod: False
      --> tuxml_version: pre-alpha v0.2
    ==> hardware
      --> cpu: Intel(R) Xeon(R) CPU           E5405  @ 2.00GHz
      --> arch: x86_64
      --> cpu_cores: 8
      --> cpu_freq: 1999
      --> ram: 8178820
[*] 2018-01-09 19:44:00 | Finding package manager
[+] 2018-01-09 19:44:00 | Package manager is apt-get
[*] 2018-01-09 19:44:00 | Updating packages repositories
[+] 2018-01-09 19:44:16 | Packages repositories updated
[*] 2018-01-09 19:44:16 | Installing default dependencies
[*] 2018-01-09 19:44:16 | Installing packages : gcc make binutils util-linux kmod e2f
sprogs jfsutils xfsprogs btrfs-progs pcmciautils ppp grub iptables openssl bc reiserf
sprogs squashfs-tools quotatool nfs-kernel-server procps mcelog libcrypto++6 apt-util
s
[+] 2018-01-09 19:45:27 | All the packages were found and installed
[*] 2018-01-09 19:45:27 | Compilation in progress
[*] 2018-01-09 19:45:41 | Compilation failed, exit status : 2
[*] 2018-01-09 19:45:41 | Analyzing error log file
[+] 2018-01-09 19:45:41 | Missing file(s)/package(s) found
[#] 2018-01-09 19:45:41 | Debian based distro
[#] 2018-01-09 19:45:41 | Those files are missing :
    openssl/bio.h
[+] 2018-01-09 19:45:48 | Dependencies built
[*] 2018-01-09 19:45:48 | Installing packages : libssl-dev
[+] 2018-01-09 19:45:55 | All the packages were found and installed
[+] 2018-01-09 19:45:55 | Restarting compilation
[*] 2018-01-09 19:45:55 | Compilation in progress
[+] 2018-01-09 19:55:38 | Compilation done
[+] 2018-01-09 19:55:38 | Successfully compiled in 00:10:10
[*] 2018-01-09 19:55:38 | Sending config file and status to database
[+] 2018-01-09 19:55:39 | Successfully sent info to db
```

Scripts
tuxml.py

```
usage: tuxml.py [-h] [-v {0,1,2}] [-V] [-c NB_CORES] [-d [KCONFIG]]
                [--no-clean]
                source_path

positional arguments:
  source_path           path to the Linux source directory

optional arguments:
  -h, --help            show this help message and exit
  -v {0,1,2}, --verbose {0,1,2}
                        increase or decrease output verbosity
                          0 : quiet
                          1 : normal (default)
                          2 : chatty
  -V, --version         display TuxML version and exit
  -c NB_CORES, --cores NB_CORES
                        define the number of CPU cores to use during the
                        compilation. By default TuxML use all the availables
                        cores on the system.
  -d [KCONFIG], --debug [KCONFIG]
                        debug a given KCONFIG_SEED or KCONFIG_FILE. If no seed
                        or file are given, the script will use the existing
                        KCONFIG_FILE in the linux source directory
  --no-clean            do not erase files from previous compilations
```

Base de données

- Regrouper les résultats de compilation

Base de données

- Regrouper les résultats de compilation
- Premier essai : JHipster

Base de données

- Regrouper les résultats de compilation
- Premier essai : JHipster
- Base de données MySQL

Base de données

- Regrouper les résultats de compilation
- Premier essai : JHipster
- Base de données MySQL
- Bibliothèque `mysqlclient`

Autotest
Description

- Suite logicielle de tests
- Outil spécialisé pour le *kernel* Linux
- Produit phare de l'industrie
- Utilise dans son cas normal une architecture "maître–esclave"

Autotest
Avantages et Inconvénients

Autotest
Avantages et Inconvénients

Avantages

- Open-source, construit principalement en Python
- Très puissant
- Actions contrôlables par script
- Documentation complète. . .

Autotest
Avantages et Inconvénients

Avantages

- Open-source, construit principalement en Python
- Très puissant
- Actions contrôlables par script
- Documentation complète. . .

Inconvénients

- Code source peu commenté et très forte imbrication
- Architecture très complexe à mettre en place
- Documentation **très** verbeuse

Autotest
Architecture

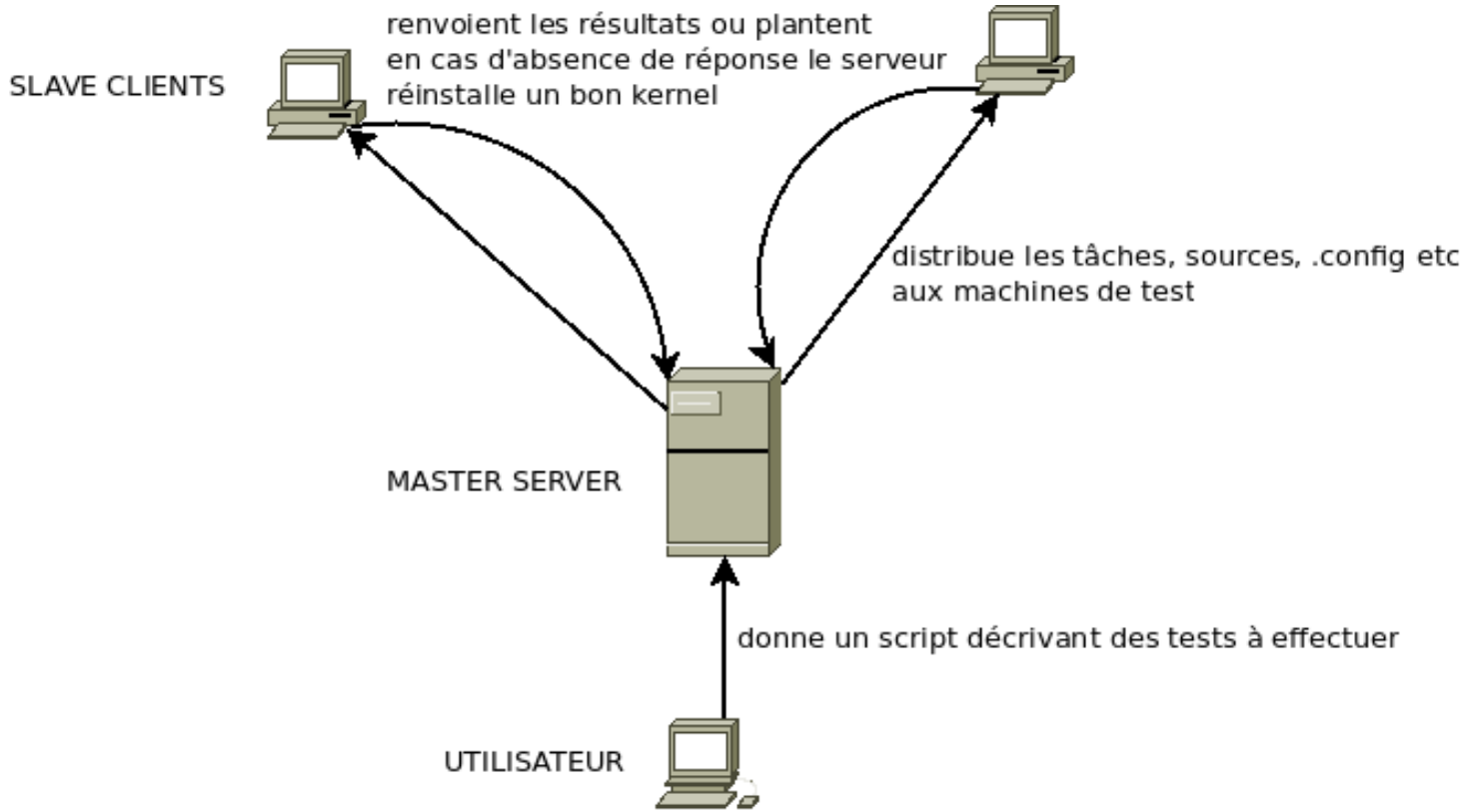


FIGURE – Architecture serveur maître et clients esclaves

Docker
Création des images

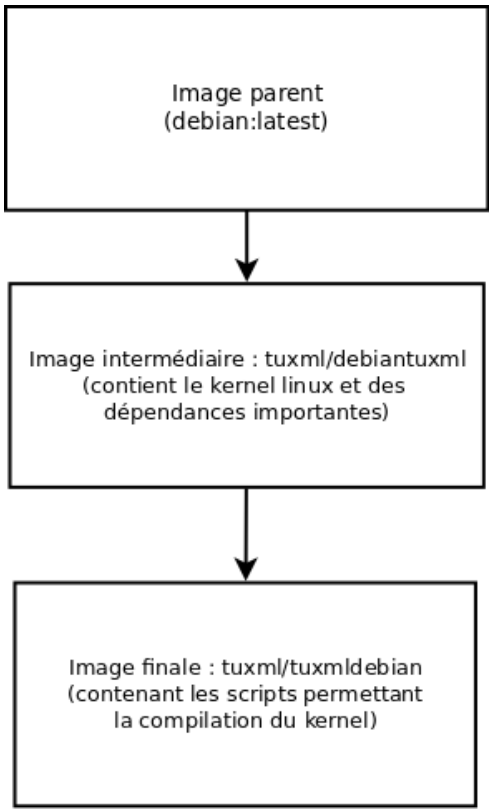
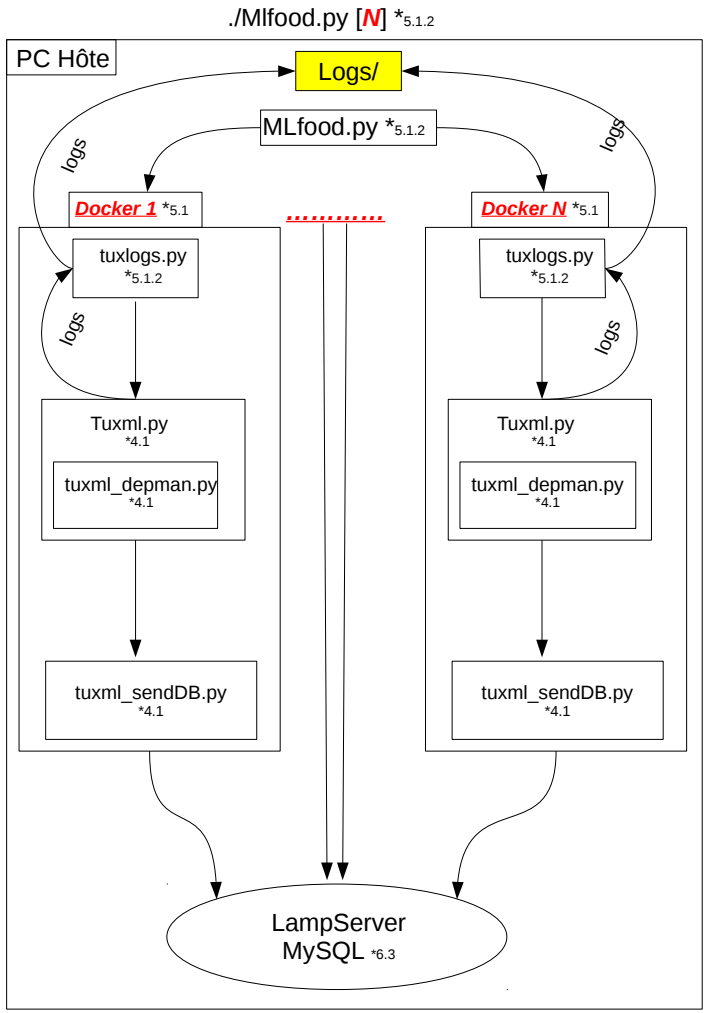


FIGURE – Schéma de création des images utilisé pour TuxML

Docker
MLfood.py



* : voir la section dans le rapport

Résultats
Entrées

- 1300 entrées sur la base de données
- Contenant actuellement les mesures de la taille du *kernel* et le temps de compilation
- Déjà des résultats préliminaires potentiellement intéressant apparaissent

Résultats
Taille du noyau

- Taille moyenne : 75.77 Mo
- Grande fourchette de taille :
  - Plus petit noyau obtenu : 12.44 Mo
  - Plus lourd : 1.793 Go !

Résultats
Temps de compilation

- Temps moyen d'environ 15 minutes
- Grande fourchette de durée :
  - Plus courte compilation sur la base : 2 minutes
  - Plus longue : $>$ 2 heures

Mise à l'échelle

- Pour pouvoir réaliser un nombre important de compilation, on souhaite utiliser des grilles de calculs :
  - Grid5000
  - IGRIDA (TuxML étant déjà compatible pour un déploiement sur cette grille)
- Plus de données, résultats plus précis
- Utilisation de Docker sur un réseau distribué de machines
- Facilité d'utilisation pour un nombre important de compilation

Approche du Machine Learning

- Étude approfondie de l'état de l'art du ML à faire
- Emploi courant de `.csv` pour le passage d'informations
- Application sur l'impact des options quand aux mesures effectuées