Plateforme Expe-Data
Application exemple et SMS

Laboratoire d'**I**nfoR**m**atique en **I**mage et **S**ystèmes d'information

LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/Ecole Centrale de Lyon

http://liris.cnrs.fr

Application log-analyzer

- Script Python (log Apache → Base de données)
- Possibilité d'utiliser une BD légère ou une BD classique
- Pas paramétrable, pas réplicable
- Mesurer les efforts nécessaires pour le rendre automatiser et rendre reproductible l'exécution du script
- Inspiration : SIGMOD Reproducibility submission of paper et webinaires Recherche reproductible

SIGMOD Reproducibility sub.

Readme for reproducibility submission of paper ID [paperID]

https://gitlab.liris.cnrs.fr/fconil/expe-data/wikis/References/SIGMOD2017-Reproducibility-readme-template.txt

- **A) Source code info**
- Repository: [url]
- Programming Language: [C/C++/java/...]
- Additional Programming Language info: [optional, e.g., java version]
- Compiler Info: [full details of compiler and version]
- Packages/Libraries Needed: [an as thorough as possible list of software packages needed]

- **B) Datasets info**
- Repository: [url]
- Data generators: [url]

- **C) Hardware Info**
- [Here you should include any details and comments about the used hardware in order to be able to accommodate the reproducibility effort. Any information about non-standard hardware should also be included. You should also include at least the following info:]
- C1) Processor (architecture, type, and number of processors/sockets)
- C2) Caches (number of levels, and size of each level)
- C3) Memory (size and speed)
- C4) Secondary Storage (type: SSD/HDD/other, size, performance: random read/sequnetial read/random write/sequnetial write)
- C5) Network (if applicable: type and bandwidth)

- **D) Experimentation Info**
- D1) Scripts and how-tos to generate all necessary data or locate datasets
  [Ideally, there is a script called: ./prepareData.sh]
- D2) Scripts and how-tos to prepare the software for system
  [Ideally, there is a script called: ./prepareSoftware.sh]
- D3) Scripts and how-tos for all experiments executed for the paper
  [Ideally, there is a script called: ./runExperiments.sh]

« Dataset »

- On ne peut pas utiliser les fichiers de log LIRIS sans les anonymiser
- Génération d'un faux fichier access.log avec un script Python existant qui se base sur des librairies de génération de données factices (faker, à remplacer par mimesis)
- Fichier résultant de 800 000 lignes (170 Mo) stocké via git lfs (Large File Storage)
  un repository git ne doit pas excéder 1 Go

Application paramétrable

- Paramétrer l'application exemple :
  - fichier de log utilisé en entrée
  - fichiers de sortie de l'application
  - nombre de lignes parsées
  - Nombre d'enregistrement traités avant de forcer un flush
  - base de données utilisée
  - Informations d'authentification à la BD

https://gitlab.liris.cnrs.fr/fconil/expe-data/wikis/Specifications/NumericalExperimentSpecifications

Application paramétrable

- Le paramétrage est possible :
  - via un fichier de configuration, le plus facile mais pas le plus flexible
  - via la ligne de commande, le plus pratique pour l'automatisation
  - via des variables d'environnement pour des données plus sensibles comme les paramètres d'authentification (https://12factor.net/config)

Les sorties de l'expérience

- Les mesures :
  - Pour log-analyzer, le temps d'exécution
  - Le fichier de sortie est au format CSV (fichier .res), facilement exploitable
- Les paramètres d'invocation et de contexte sont enregistrés dans un fichier json (fichier .inv)
- Log applicatif (fichier .log)
- Le nom des fichiers de sortie reprend les paramètres importants :

  LOG_{batch-size}_{nb-parsed-lines}_{no-run}.res
  LOG_{batch-size}_{nb-parsed-lines}_{no-run}.log
  LOG_{batch-size}_{nb-parsed-lines}_{no-run}.inv

Les sorties de l’expérience

- La formalisation des sorties et en particulier des mesures a pour but l’automatisation de l’analyse des résultats
- Ce n’est pas demandé par SIGMOD Reproducibility submission
- Popper a intégré l’analyse des résultats dans son workflow de recherche reproductible

Log applicatif

- Comme on va automatiser l'exécution à de multiples reprises de l'application, il faut garder une trace du déroulement de l'exécution
- Utilisation de la librairie standard du langage. Les langages importants ont tous ce type de librairie.
- Niveau de log paramétrable
- Sortie paramétrable : console, fichier, ...

Mauvaise idée

- Utiliser la librairie de log pour écrire le fichier de mesure au fil de l'eau
- Partie d'une mauvaise interprétation : fil de l'eau → librairie de log
- Pas du tout adapté
- Complexification du code inutile

Utiliser l'application

- Instructions d'installation, de mise en œuvre
  - Le concepteur n'a pas toujours conscience de tout ce qu'il faut mettre en oeuvre
- Scripts de lancement
  - Nécessaire dès que l'on a plusieurs paramètres en ligne de commande
  - Peut montrer comment exécuter l'application
- Coder pour plusieurs OS a un coût
- Intérêt d'un système de virtualisation
  - Préserve la machine de développement
  - Peut éviter un développement multi-OS
  - Permet d'expliciter l'installation de l'application et sa mise en oeuvre

Automatisation - Popper

- Outil permettant d'appliquer des principes d'intégration continue à l'exécution d'expériences scientifiques
- L’application doit fournir :
  - un script de **build**
  - un script de **run**
  - un script de **check**
  - un script d’**analyse**
- Popper repose principalement sur la conteneurisation docker
- Malheureusement, Popper tenait plus du prototype au moment des tests et nous avons rencontré trop de difficultés dans sa mise en œuvre pour que nous puissions le considérer comme un outil

Automatisation – Runner simple

- le "runner", run.py, récupère les paramètres permettant de déterminer les expériences atomiques à exécuter
- ces paramètres sont stockés dans un fichier de configuration, run.conf
- le "runner" est en charge de construire la base du nom des fichiers résultats à partir des paramètres de lancement : LOG_{taille-buffer}_{nb-lignes}_{no-run}
- l'application va, pour chaque lancement, produire un fichier json avec les paramètres avec lesquels elle a été invoquée (d'où l'extension .inv)

```
$ cat run.conf

[global]
experience_path =
results_path =
access_log_filename =
access_log_20170907.log
batch_sizes = 1, 10, 100, 1000,
10000
max_lines = 100, 1000, 10000,
100000, 800000
nb_runs = 5

[SQLite]

[MySQL]
```

Automatisation – Analyse

- Le script d'analyse, analyse.py, reprend la configuration de l'ensemble des expériences, run.conf, pour produire quelques graphiques
- On veut voir si le temps d'exécution évolue linéairement avec le nombre de lignes à parser et voir l'impact du buffer d'écriture en base de données
- Le script Python utilise les librairies :
  - **Pandas** pour la manipulation des fichiers de données csv
  - **SciPy** pour le calcul de la régression linéaire et de ses caractéristiques
  - **matplotlib** pour la génération des graphiques au format png ou svg

Virtualisation - Docker

- Le "runner" simple implique que l'utilisateur installe l'environnement d'exécution sur sa machine
- docker présente plusieurs avantages :
  - l'image de base créée est non modifiable sans action spécifique, le container créé à partir de cette image pour lancer l'expérience est censé être identique, ce qui est favorable à la reproductibilité
  - les images sont plus légères que les machines virtuelles et démarrent plus vite
  - la création d'image à partir d'un fichier de description, le Dockerfile est assez intuitive
  - une fois les images mises au point, on peut envisager de les utiliser dans un système d'intégration continue (GitLab CI/CD)

Contraintes Docker

- docker a pour but de mettre en place des micros-services : **à un container doit correspondre un exécutable**, ce qui oblige les développeurs à créer plusieurs conteneurs qu'il faut "orchestrer"
- l'image construite est figée et on ne veut pas y stocker des **éléments d'authentification**
- le système de cache fait correspondre une chaîne de caractères du Dockerfile avec un élément de cache (RUN apt-get update)

Mise en œuvre de Docker

- Le repository de l'application logs-analyzer est privé, il faut s'authentifier pour cloner le repository et on ne veut pas que les éléments d'authentification soient stockés dans l'image
- Pour le moment, on n'utilise pas docker swarm et on n'a donc pas de fonctionnalité de gestion des secrets.
- un script shell build-image.sh récupère le fichier des packages Python requis dans le repository cloné et le copie dans le contexte du Dockerfile pour « l'injecter » dans l'image générée
- Le script run-image.sh crée le container à partir de l'image générée et monte le repository logs-analyzer en volume. Il copie au préalable le fichier de configuration spécifique run.conf dans le répertoire du "runner".
- Avantage :
  - on peut modifier le code de l'application dynamiquement et on n'a pas besoin de regénérer l'image chaque fois que l'on veut changer des paramètres ou modifier le comportement de l'application
- Inconvénients :
  - Docker s'exécute en tant que root et les fichiers qu'il crée dans le volume appartiennent donc à root

MySQL - Docker compose

- 2 containers :
  - container MySQ
  - container avec log-analyzer
- La BD n'est pas prête lorsque le container de l'application est lancé
- Modification de l'application pour qu'elle tente plusieurs connexions à la BD avant de commencer le traitement

https://docs.docker.com/compose/startup-order/

The problem of waiting for a database to be ready is really just a subset of a much larger problem of distributed systems. In production, your database could become unavailable or move hosts at any time. Your application needs to be resilient to these types of failures.

```
services:
  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: whatever
      MYSQL_DATABASE: mysql
      MYSQL_USER: experunner
      MYSQL_PASSWORD: doitagain
    volumes:
      - "db_data:/var/lib/mysql"
      -
"./docker_ep/init.sql:/docker-
entrypoint-initdb.d/init.sql"

  app:
    depends_on:
      - db
    build: app
    volumes:
      - "${LOGS_ANALYZER_WD}:/app"
    environment:
      LA_LOGIN: experunner
      LA_PASSWORD: doitagain
    restart: on-failure

volumes:
  db_data:
```

Virtualisation - Vagrant

- Vagrant est un outil pour construire et gérer des machines virtuelles. L'outil était initialement une surcouche à VboxManage (VirtualBox), il supporte maintenant VMware et Docker.
- Permet de construire une seule machine virtuelle pour la BD et l'application
- Tout comme Docker héberge des images sur son hub, Vagrant héberge des box sur Vagrant Cloud
- On crée une machine virtuelle via un VagrantFile (syntaxe ruby) et on peut installer et configurer les applications avec différents outils de provisionning : shell, Puppet, Chef, Ansible, ...

Virtualisation - Vagrant

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "debian/jessie64"

  config.vm.synced_folder "..", "/app"

  config.vm.provision :shell, path: "build.sh"
end
```

```
#!/usr/bin/env bash
echo -e "\n--- Updating packages list ---\n"
apt-get update
echo -e "\n--- Install SQLite package ---\n"
apt-get install -y sqlite3

DBPASSWD="whatever"
DB_LA_LOGIN="experunner"
DB_LA_PASSWORD="doitagain"

debconf-set-selections <<< "mysql-server mysql-server/root_password password $DBPASSWD"
debconf-set-selections <<< "mysql-server mysql-server/root_password_again password
$DBPASSWD"

apt-get install -y mysql-server mysql-client
...
```

Vagrant – shared folders

- Par homogénéité avec ce qui a été fait avec Docker, le code de l'application est sur la machine hôte dans un "shared folder" /app.
- Cela nécessite l'installation de VirtualBox Guest Additions sur le système invité, maintenant automatisé par un plugin vagrant-vbguest.
- **Avantages** :
- les fichiers créés dans le shared folder à partir de la machine virtuelle appartiennent à l'utilisateur qui lance Vagrant, et non à root
- les modifications faites sur les fichiers des Shared folders sont immédiatement utilisables dans la machine virtuelle (pas de redémarrage nécessaire)
- **Inconvénients** :
- il n'y a pas d'exécutable automatiquement lancé au démarrage de la machine virtuelle. Une fois la machine virtuelle démarrée par **vagrant up**, nous nous sommes connectés via **vagrant ssh** puis nous avons lancé les expériences manuellement en se plaçant dans le répertoire "runner" et en exécutant **python3 run.py run.conf**.
- Contrairement à Docker, la machine virtuelle évolue lorsqu'on la manipule et lorsqu'on la relance elle n'est plus dans l'état initial dans lequel elle se trouvait juste après la construction de l'environnement d'exécution.

Impact des env. d’exécution

- Exécution de :
  - 5 run
  - pour 100 000 ligne à parser
  - avec un flush toutes les 1000 lignes

Impact des env. - sqlite

- CPU time (en secondes)

donnée	**local**	**docker**	**vagrant**
moyenne	26.828691	30.579208	38.113468
écart-type	0.556357	0.409357	3.271832
min	26.388133	30.175282	32.418673
max	27.760235	31.041574	40.403848

- Global time (en secondes)

donnée	**local**	**docker**	**vagrant**
moyenne	27.586798	31.355139	42.613863
écart-type	0.650764	0.380206	2.714536
min	27.111030	30.928814	37.974621
max	28.698819	31.862604	44.624035

Impact des env. - mysql

- CPU time (en secondes)

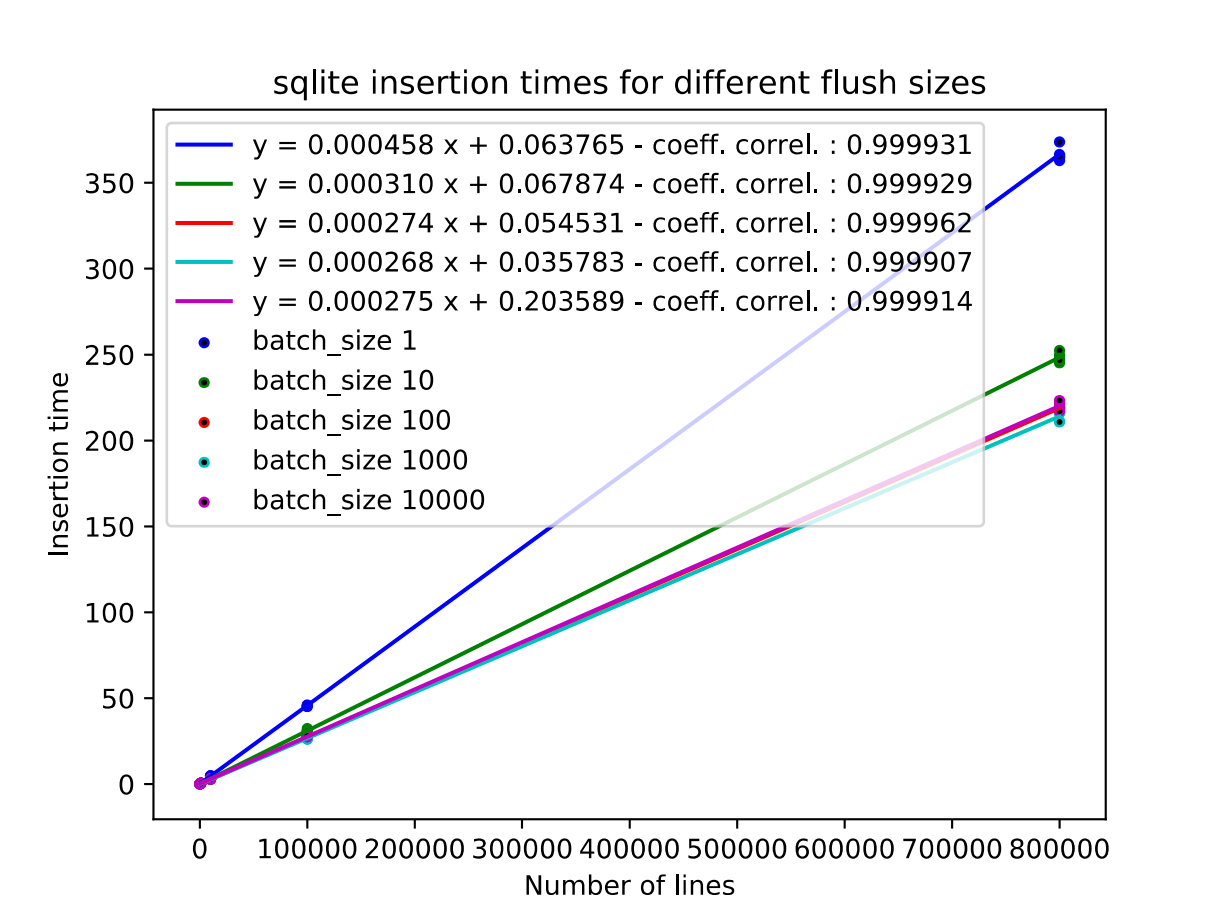
donnée	**local**	**docker**	**vagrant**
moyenne	51.622526	55.362695	50.048604
écart-type	1.307686	3.392192	0.472691
min	50.440980	51.801245	49.521434
max	53.831369	59.708447	50.547458

- Global time (en secondes)

donnée	**local**	**docker**	**vagrant**
moyenne	68.889512	78.011632	68.110050
écart-type	1.385506	16.809095	1.366952
min	67.857578	67.741459	66.656107
max	71.278008	107.597180	69.776472

Pour docker, le temps maximum en global time est celui du premier lancement où l'application doit attendre que le serveur MySQL soit prêt pour commencer.

Sortie graphique SQLite (local)



Sortie graphique MySQL (local)