

Gestion efficace de calculs numériques

exemple avec Execo-Engine

Laurent Pouilloux

Laboratoire de Mécanique des Fluides et d'Acoustique

Ateliers et Séminaires Pour l'Informatique et le Calcul Scientifique
Vendredi 10 mars 2017

Table des matières

1 Introduction

2 Execo

3 Execo-engine

4 Exemples

Contexte

Des besoins croissants en calcul

- modélisation numérique
- traitement de données expérimentales

Contexte

Des besoins croissants en calcul

- modélisation numérique
- traitement de données expérimentales

Des moyens variés à votre disposition

- votre station de travail ou votre portable
- des clusters locaux* et régionaux
- des moyens nationaux et internationaux

* voir les cours et TP Newton sur

<http://lmfa.ec-lyon.fr/spip.php?article833>

Principe général du fonctionnement des calculateurs

Des utilisateurs



besoin en cœurs, mémoire, pour
une durée déterminée

Principe général du fonctionnement des calculateurs

Des utilisateurs



Des ressources de calcul



des cœurs, de la mémoire, des GPU, ...

Principe général du fonctionnement des calculateurs

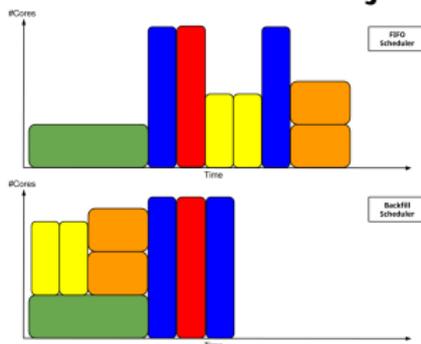
Des utilisateurs



Des ressources de calcul



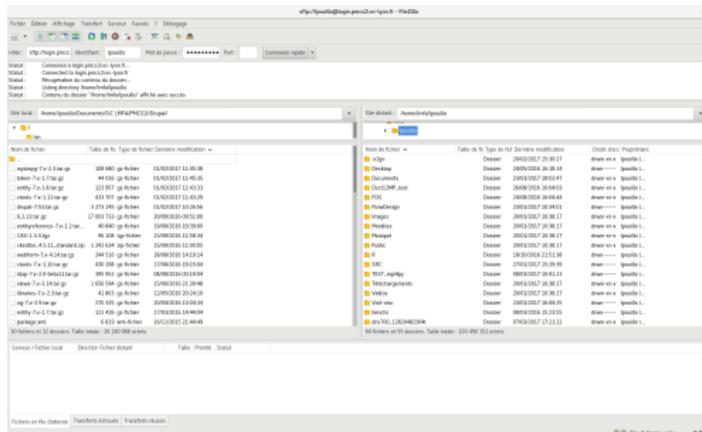
Un ordonnanceur de job



Optimise l'utilisation des ressources

Comment y accède-t-on ?

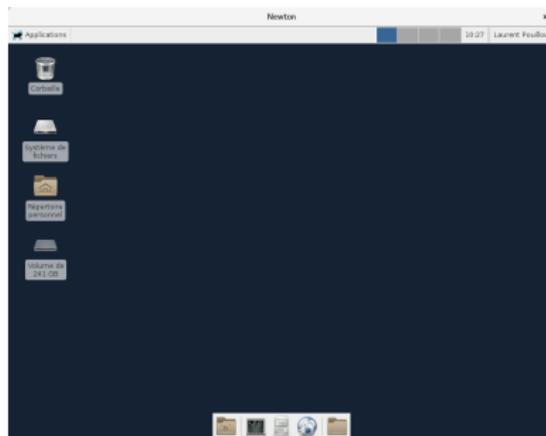
- connexion SFTP



FileZilla

Comment y accède-t-on ?

- connexion SFTP
- bureau graphique



X2go

Exécution de code sur les machines

Sessions interactives

Exécution à la main sur les calculateurs

- attente de la disponibilité des ressources
- source d'erreurs
- non reproductible

Exécution de code sur les machines

Sessions interactives

Exécution à la main sur les calculateurs

- attente de la disponibilité des ressources
- source d'erreurs
- non reproductible

Soumission de jobs

Écriture d'un script permettant l'exécution du code

- gestion manuelle des dépendances entre jobs
- pénible si nombreux jobs à lancer

Exécution de code sur les machines

Sessions interactives

Exécution à la main sur les calculateurs

- attente de la disponibilité des ressources
- source d'erreurs
- non reproductible

Soumission de jobs

Écriture d'un script permettant l'exécution du code

- gestion manuelle des dépendances entre jobs
- pénible si nombreux jobs à lancer

Scripting bash

Création et soumission automatique de jobs

- syntaxe limitée
- quasi-obligatoire de travailler sur la frontale

Utilisation de moteurs d'expériences

Pourquoi faire ?

- diagramme de régime
- tests de scaling
- traitement distribué de données

Utilisation de moteurs d'expériences

Pourquoi faire ?

- diagramme de régime
- tests de scaling
- traitement distribué de données

Comment faire ?

- définition d'un plan d'expérience
- soumission automatisée sur les ressources de calcul
- récupération des résultats
- visualisation
- analyse statistique
- évolution automatique de l'expérience

Utilisation de moteurs d'expériences

Pourquoi faire ?

- diagramme de régime
- tests de scaling
- traitement distribué de données

Comment faire ?

- définition d'un plan d'expérience
- soumission automatisée sur les ressources de calcul
- récupération des résultats
- visualisation
- analyse statistique
- évolution automatique de l'expérience

Mise en place d'un **cahier de manip** numérique

La solution Execo



Bibliothèque Python (ASPICS PYTHON - 03/02/2017)

- la gestion fine des processus Unix
⇒ ASPICS SHELL - 07/04/2017
- l'interaction avec un calculateur
⇒ ASPICS NEWTON - 06/02/2017
- la réalisation de moteurs d'expériences numériques
⇒ ASPICS EXECO ENGINE - **Aujourd'hui**

La solution Execo



Bibliothèque Python (ASPICS PYTHON - 03/02/2017)

- la gestion fine des processus Unix
⇒ ASPICS SHELL - 07/04/2017
- l'interaction avec un calculateur
⇒ ASPICS NEWTON - 06/02/2017
- la réalisation de moteurs d'expériences numériques
⇒ ASPICS EXECO ENGINE - **Aujourd'hui**

Développé par M. Imbert (SED Inria Rhône-Alpes) sous Licence GPLv3.

<http://execo.gforge.inria.fr>

Table des matières

- 1 Introduction
- 2 Execo
- 3 Execo-engine
- 4 Exemples

Installation d'Execo

- Fonctionne sur Linux et Mac (*Windows* \Rightarrow *VM ou cygwin*)
- Compatible Python 2 ou 3
- Disponible dans le dépôt PyPI
`pip install --user execo`
- Utilisable interactivement ou via des scripts

Gestion des processus locaux

Process

- contrôle de l'exécution : `start`, `wait`, `kill`
- informations sur l'état : `error`, `exit_code`
- entrées/sorties : `stdout`, `stderr`, `stdin`

Gestion des processus locaux

Process

- contrôle de l'exécution : `start`, `wait`, `kill`
- informations sur l'état : `error`, `exit_code`
- entrées/sorties : `stdout`, `stderr`, `stdin`

```
In [1]: from execo import Process
In [2]: test = Process('ls').run()
In [3]: print test.exit_code
0
In [4]: print test.stdout
Backfill.png
computer-user-icon-27.png
computer-user-icon-5.png
Curie-GENCI-606x513.jpg
execo.png
..
```

Gestion des processus distants

SshProcess

- Héritage des méthodes de **Process**
- Cible : un hôte distant (hostname, user, port, keyfile)

Gestion des processus distants

SshProcess

- Héritage des méthodes de **Process**
- Cible : un hôte distant (hostname, user, port, keyfile)

```
In [1]: from execo import SshProcess
```

```
In [2]: test = SshProcess('ls /tmp', 'newton').run()
```

```
In [3]: print test.stdout
```

```
systemd-private-5cc33162c1a94b699ce99258a35a59ad-munin-node.service-QrB
```

```
systemd-private-5cc33162c1a94b699ce99258a35a59ad-ntpd.service-5GAAP0
```

```
tmux-5544
```

```
In [4]: print test.host.address
```

```
newton
```

Remote ⇒ exécution parallèle sur différents hôtes

Transfert de fichiers

- **Put**, envoyer les fichiers sur une ou plusieurs machines
- **Get**, récupérer les données une fois l'exécution terminée

Transfert de fichiers

- **Put**, envoyer les fichiers sur une ou plusieurs machines
- **Get**, récupérer les données une fois l'exécution terminée

Peut être utilisé de manière *asynchrone*, i.e. lancer le transfert et faire autre chose en attendant

Gestion des logs

- logger Python standard
- configuration par défaut :
 - silencieux
 - sauf si erreurs d'exécution
 - hautement configurable
- facilite l'analyse post-mortem

Table des matières

- 1 Introduction
- 2 Execo
- 3 Execo-engine**
- 4 Exemples

Implémentation du cycle de vie de la campagne

Engine

- gestion centralisée des options et arguments
- création d'un répertoire pour l'expérience
- permet un restart aisé

```
class MyEngine(Engine):  
    def __init__(self):  
        <INITIALIZATION>  
  
    def run(self):  
        """ """  
        <EXPERIMENTAL WORKFLOW>
```

Exploration automatisée de gamme de paramètres

Génération des combinaisons à traiter avec **sweep**

```
In [18]: parameters = {"Re": [10**i for i in range(1,5)],
...:                  "Pr": [10**i for i in range(-5,5,2)],
...:                  "BC": ['free-slip', 'no-slip']}
{'BC': ['free-slip', 'no-slip'],
 'Pr': [1e-05, 0.001, 0.1, 10, 1000],
 'Re': [10, 100, 1000, 10000]}
```

```
In [19]: sweep(parameters)
```

```
Out[19]:
```

```
{'BC': 'free-slip', 'Pr': 1e-05, 'Re': 10},
{'BC': 'no-slip', 'Pr': 1e-05, 'Re': 10},
{'BC': 'free-slip', 'Pr': 1e-05, 'Re': 100},
{'BC': 'no-slip', 'Pr': 1e-05, 'Re': 100},
{'BC': 'free-slip', 'Pr': 1e-05, 'Re': 1000},
{'BC': 'no-slip', 'Pr': 1e-05, 'Re': 1000},
{'BC': 'free-slip', 'Pr': 1e-05, 'Re': 10000},
...
```

Création d'un itérateur

ParamSweeper : objet permettant le balayage simplifié des paramètres

```
In [23]: sweeper = ParamSweeper('test', sweeps)
```

```
In [24]: ls test
```

```
done inprogress
```

- création d'un répertoire pour stocker l'état des combinaisons
- itération via l'objet sweeper
- méthodes utiles :
 - Récupérer la combinaison suivante :
`comb = sweeper.get_next()`
 - Ignorer une combinaison :
`comb = sweeper.skip(comb)`
 - Marquer une combinaison comme finie : `sweeper.done(comb)`
ou à refaire : `sweeper.cancel(comb)`
 - Redéfinir les combinaisons :
`sweeper.set_sweeps(new_sweeps)`

Création d'un itérateur

ParamSweeper : objet permettant le balayage simplifié des paramètres

```
In [23]: sweeper = ParamSweeper('test', sweeps)
```

```
In [24]: ls test
```

```
done inprogress
```

- création d'un répertoire pour stocker l'état des combinaisons
- itération via l'objet sweeper
- méthodes utiles :
 - Récupérer la combinaison suivante :
`comb = sweeper.get_next()`
 - Ignorer une combinaison :
`comb = sweeper.skip(comb)`
 - Marquer une combinaison comme finie : `sweeper.done(comb)`
ou à refaire : `sweeper.cancel(comb)`
 - Redéfinir les combinaisons :
`sweeper.set_sweeps(new_sweeps)`

Avantages majeurs : simplification de la gestion du cycle de vie de l'expérience, possibilité de reprise, accès en parallèle

Table des matières

- 1 Introduction
- 2 Execo
- 3 Execo-engine
- 4 Exemples**

Un moteur d'expérience *local*

Lancement depuis sa station, exécution locale

Fire Dynamics Simulator : LES d'un canal avec source de chaleur

Un moteur d'expérience *local*

Lancement depuis sa station, exécution locale

Fire Dynamics Simulator : LES d'un canal avec source de chaleur

Paramètres

- Nombre Pr : 0.1, 0.71, 100, 200
- Nombre de points : 16, 32, 64

Un moteur d'expérience *local*

Lancement depuis sa station, exécution locale

Fire Dynamics Simulator : LES d'un canal avec source de chaleur

Paramètres

- Nombre Pr : 0.1, 0.71, 100, 200
- Nombre de points : 16, 32, 64

Plan d'expérience

- 1 Création d'un itérateur sur les paramètres
- 2 Génération d'un fichier d'input
- 3 Exécution du code
- 4 Visualisation

Un moteur d'expérience *local*

Lancement depuis sa station, exécution locale

Fire Dynamics Simulator : LES d'un canal avec source de chaleur

Paramètres

- Nombre Pr : 0.1, 0.71, 100, 200
- Nombre de points : 16, 32, 64

Plan d'expérience

- 1 Création d'un itérateur sur les paramètres
- 2 Génération d'un fichier d'input
- 3 Exécution du code
- 4 Visualisation

Possibilité d'ajouter des valeurs aux paramètres sans tout refaire !

Précision et scaling du calcul de π par Monte-Carlo

Lancement depuis sa station, exécution sur Newton

Paramètres

- nombre de points : 1000000, 10000000, 100000000, 1000000000
- nombre de coeurs : 1, 2, 4, 8, 16
- version de Python: system, 2.7.10_intel_2015a, 2.7.11_foss_2016a

Total : 60 combinaisons

Précision et scaling du calcul de π par Monte-Carlo

Lancement depuis sa station, exécution sur Newton

Paramètres

- nombre de points : 1000000, 10000000, 100000000, 1000000000
- nombre de coeurs : 1, 2, 4, 8, 16
- version de Python: system, 2.7.10_intel_2015a, 2.7.11_foss_2016a

Total : 60 combinaisons

Plan d'expérience

- 1 création d'un itérateur sur les paramètres
- 2 pour chaque combinaison :
 - création d'un répertoire sur Newton
 - génération d'un fichier SLURM et upload sur Newton
 - soumission du job sur la machine compil
 - attente de la fin de l'exécution
 - récupération des fichiers de résultats en local
- 3 production des figures

Gestion de jobs chaînés

- définition d'un template et d'un nombre de jobs à chaîner
- ajout automatique de la dépendance au fichier batch
- soumission séquentielle des jobs

Post traitement de DNS à l'IDRIS

ANR Stratimix

Plan d'expérience

- 1 génération d'un espace de paramètres à partir des runs
- 2 pour chaque combinaison
 - création d'un répertoire sur Ada
 - génération d'un fichier LL
 - soumission du job
 - attente de la fin de l'exécution
 - récupération des fichiers de résultats en local

Pour aller plus loin

- soumission des jobs en parallèle
- ajout d'une phase de compilation
- utilisation de différents clusters simultanément
- mise à jour des combinaisons à traiter
- ...

Execo : Un cahier de manip pour le numérique ?

- adaptable à de nombreux calculateurs
- gestion simplifiée des campagnes de jobs
- possibilité de gestion asynchrone des processus
- log horodaté de l'exécution globale
- couplé à du versionning, permet de rejouer des anciens codes

Execo : Un cahier de manip pour le numérique ?

- adaptable à de nombreux calculateurs
- gestion simplifiée des campagnes de jobs
- possibilité de gestion asynchrone des processus
- log horodaté de l'exécution globale
- couplé à du versionning, permet de rejouer des anciens codes

Permet de faciliter la reproductibilité expérimentale

Execo : Un cahier de manip pour le numérique ?

- adaptable à de nombreux calculateurs
- gestion simplifiée des campagnes de jobs
- possibilité de gestion asynchrone des processus
- log horodaté de l'exécution globale
- couplé à du versionning, permet de rejouer des anciens codes

Permet de faciliter la reproductibilité expérimentale

Autres solutions

- GC3Pie :
<https://github.com/uzh/gc3pie>
- OpenMOLE :
<https://www.openmole.org/>
- Sumatra :
<https://pythonhosted.org/Sumatra/>