Execo

Réalisation de campagnes de calcul automatisées

Laurent Pouilloux[1] - Matthieu Imbert[2]

Journée ExpéNum - mardi 22 octobre 2018

1. CNRS - Laboratoire de Mécanique des Fluides et d'Acoustique - EC Lyon
2. INRIA - Laboratoire de l'Informatique du Parallélisme - ENS Lyon

Introduction

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

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

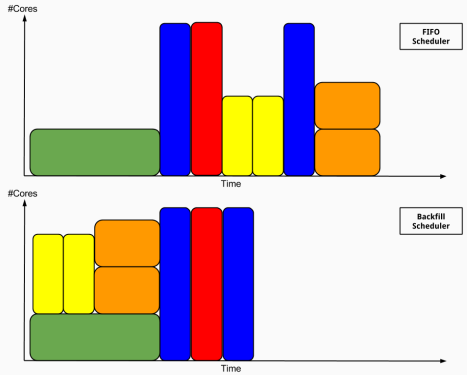
des cœurs, de la mémoire, des GPU, ...

Principe général du fonctionnement des calculateurs

Des utilisateurs

Des ressources de calcul

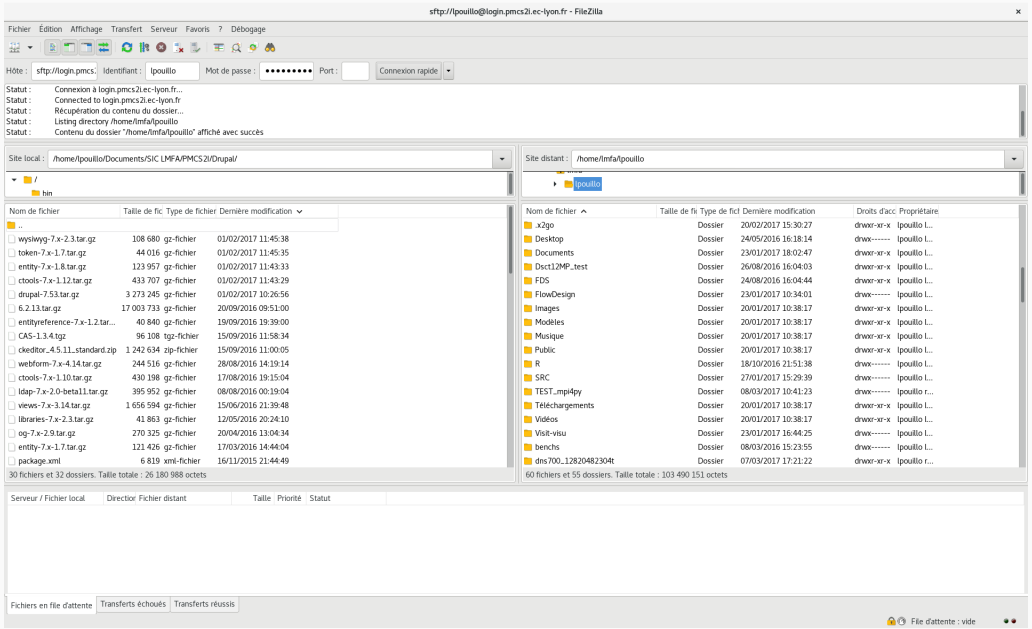
Un ordonnanceur de job



Optimise l'utilisation des resources

Comment y accède-t-on ?

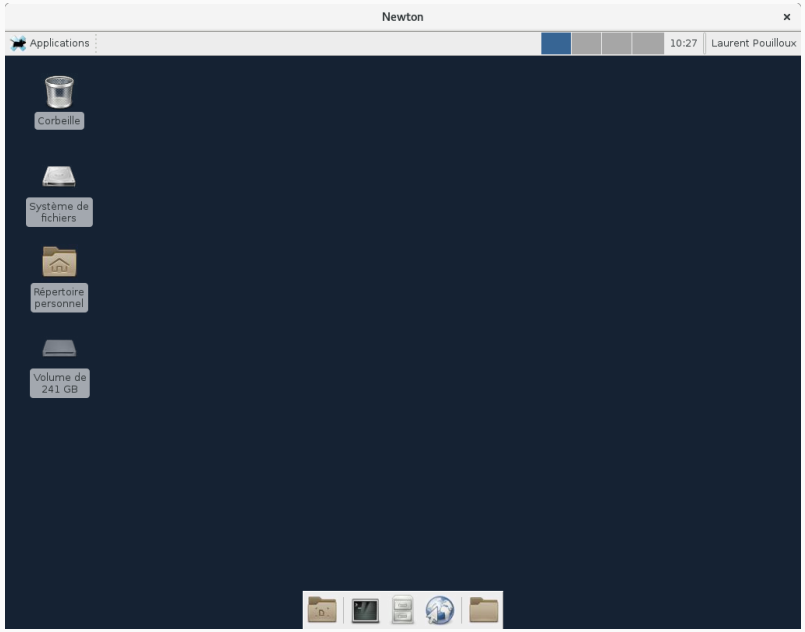
- connexion SFTP



FileZilla

Comment y accède-t-on ?

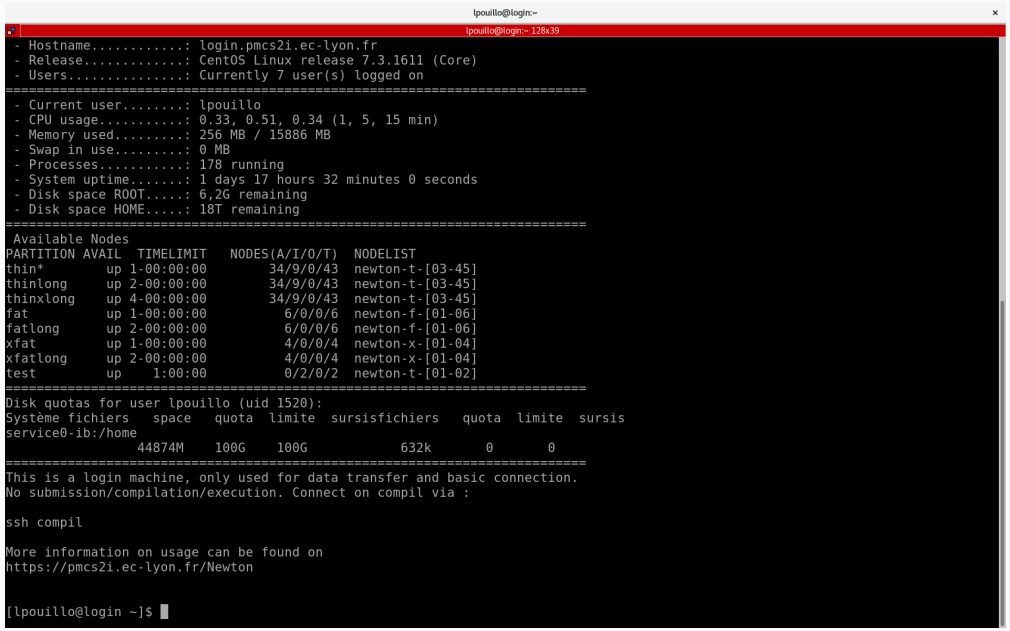
- connexion SFTP
- bureau graphique



X2go

Comment y accède-t-on ?

- connexion SFTP
- bureau graphique
- **accès via SSH**



console ssh

Exécution de code sur les machines

Sessions interactives
Exécution à la main sur les calculateurs

- attente de la disponibilités 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 disponibilités 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 disponibilités 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 resources de calcul
- récupération des résultats
- visualisation et analyse statistique
- évolution automatique de l'expérience

Utilisation de moteurs d'expériences

Pourquoi faire ?

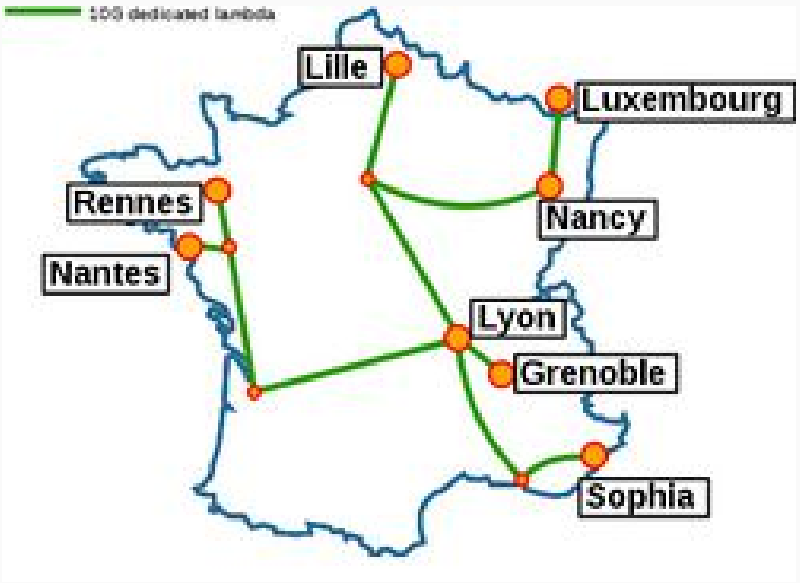
- diagramme de régime
- tests de scaling
- traitement distribué de données

Comment faire ?

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

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

Expérimentations sur Grid'5000



- architecture multi-sites interconnectés
- variabilité des calculateurs disponibles
- intéraction avec les systèmes de réservations de ressources
- mise en place d'un environnemnent pour l'expérience

La solution Execo

API Python permettant de contrôler des **processus unix locaux ou distants**, et de scripter des expériences de calcul numérique

La solution Execo

API Python permettant de contrôler des **processus unix locaux ou distants**, et de scripter des expériences de calcul numérique

- facile, rapide et intuitif. On écrit le script comme on le pense
- contrôle à grain fin, *e.g.* facile de récupérer d'un seul coup stdout, stderr, exit code, ...
- asynchrone, *e.g.* on démarre un process A, puis le B, on attend B, on tue A
- optimisé et scalable : permet de gérer de manière parallèle des milliers de processus distants
- système de log efficace :
  - configuration par défaut donnant ce qu'il faut comme sortie
  - possibilité d'analyser en direct ou post-mortem analysis le workflow
- mécanisme de transfert de fichiers efficace

Execo

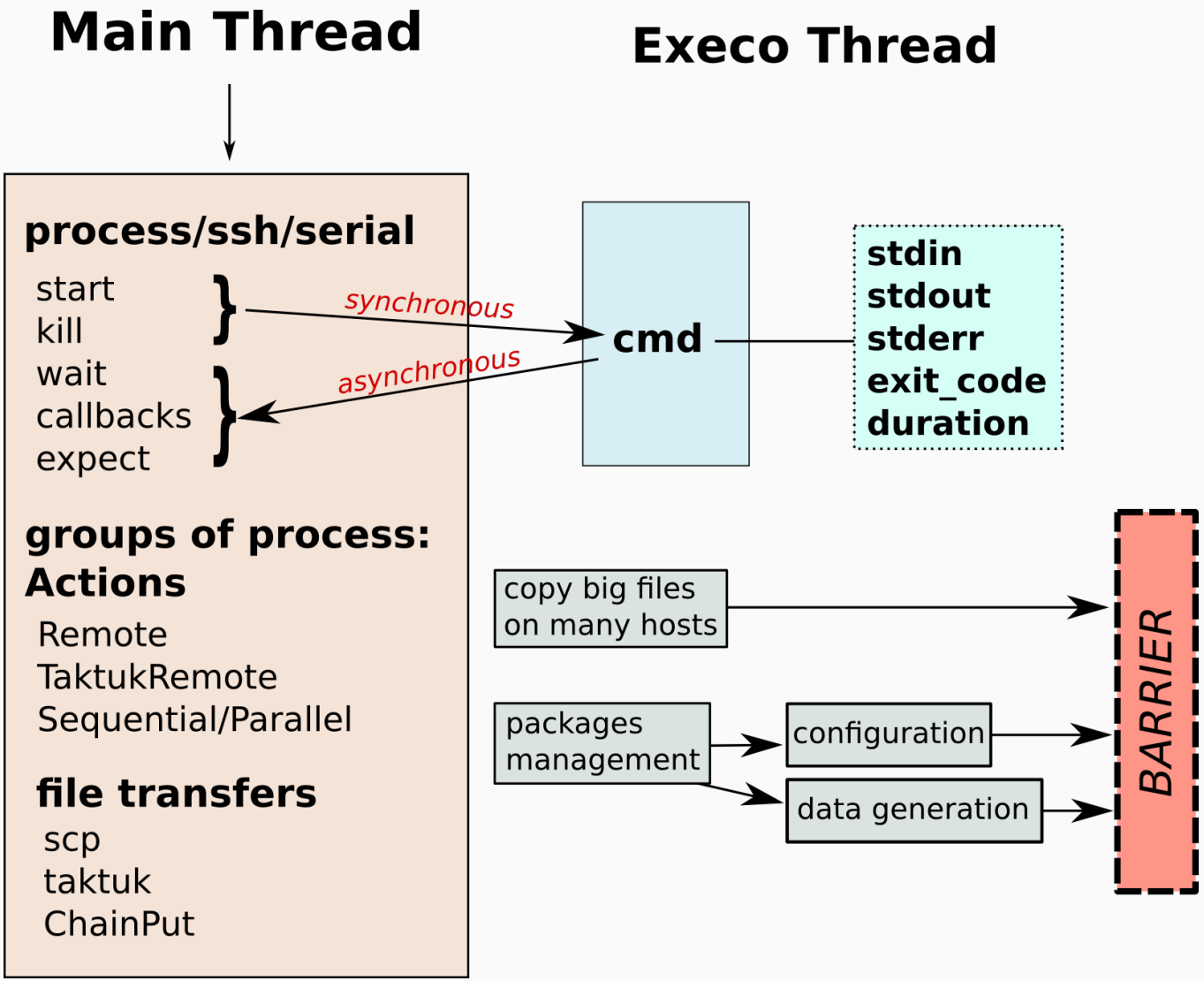
Caractéristiques d'Execo

processus unix
Repose sur le forking ou des outils ssh pour exécuter des processus

transferts de fichiers
Permet d'utiliser des scp parallèles, taktuk, ou un système de broadcast chainé très efficace.

moteur d'expérience
outils pour simplifier le développement de cahier de manip numériques : exploration d'un espace de paramètres avec un itérateur persistent, classe dédiée aux moter d'expériences

Vue générale d'Execo



Installation d'Execo

- Fonctionne sur Linux / Mac / Windows (via le sous-système Linux)
- Compatible Python 2 ou 3
- Disponible dans le dépot 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.serv
systemd-private-5cc33162c1a94b699ce99258a35a59ad-ntpd.service-5G
tmux-5544
In [4]: print test.host.address
newton
```

Remote $\Rightarrow$ 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

Execo-engine

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

Exemples

Un moteur d'expérience *local*

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

Un moteur d'expérience *local*

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*

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*

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 !

Execution d'un benchmark MPI sur différents clusters Grid'5000

https://www.grid5000.fr/mediawiki/index.php/Execo_Practical_Session

Précision et scaling du calcul de π par Monte-Carlo

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

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 dépendence au fichier batch
- soumission séquentielle des jobs

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-Engine : une toolbox pour faire des cahiers de manips

- 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-Engine : une toolbox pour faire des cahiers de manips

- 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-Engine : une toolbox pour faire des cahiers de manips

- 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

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