ÉCOLE CENTRALE LYON

FLMSN @ Lyon

cnrs

Execo

Réalisation de campagnes de calcul automatisées

Laurent Pouilloux

Journée Reproductibilité ARAMIS - jeudi 23 mai 2019

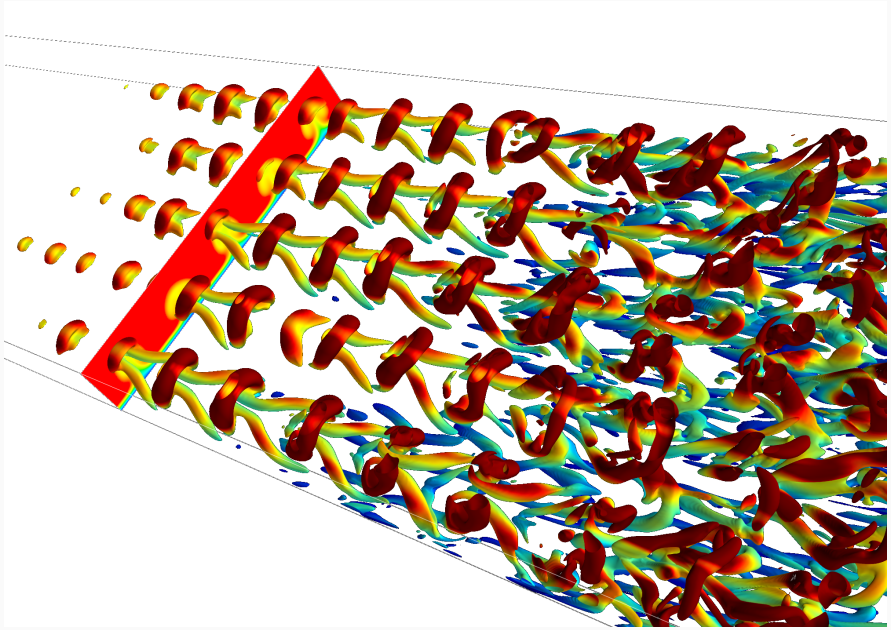
CNRS - Laboratoire de Mécanique des Fluides et d'Acoustique - EC Lyon

Introduction

Contexte

Des besoins croissants en calcul

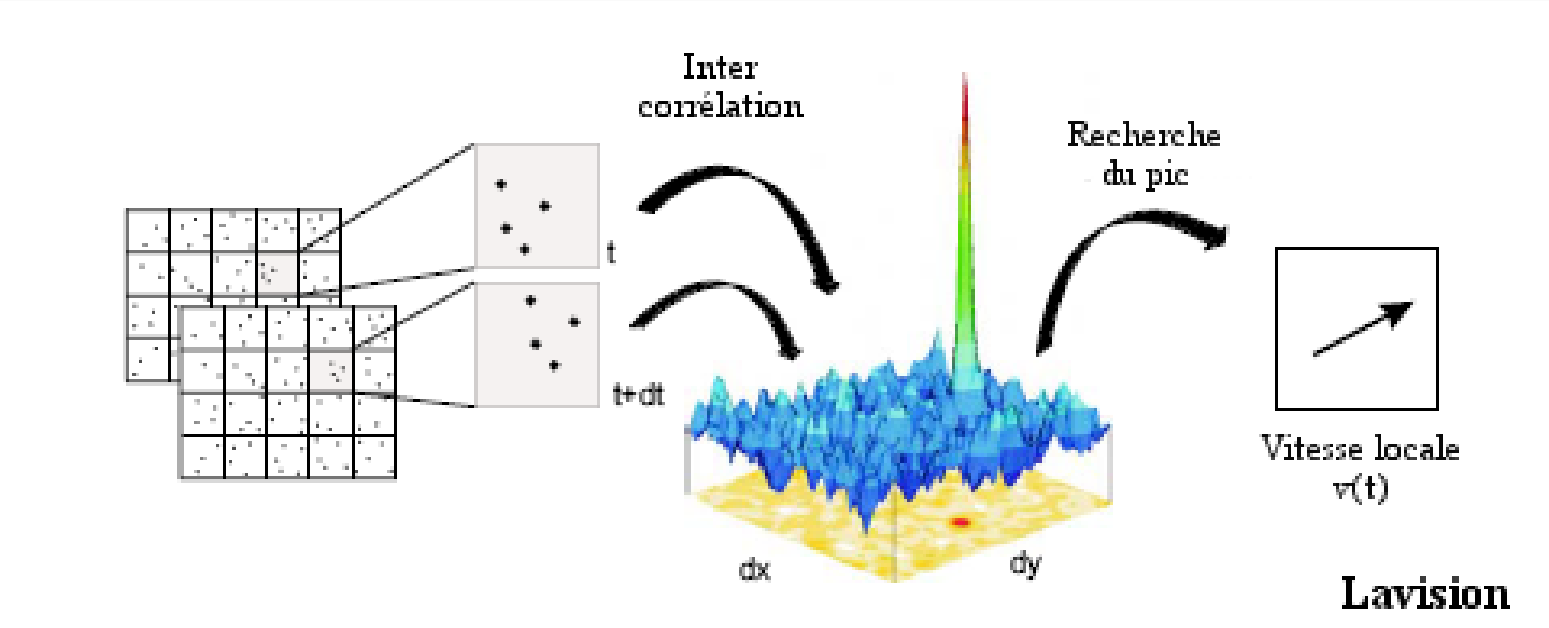
- modélisation numérique



Contexte

Des besoins croissants en calcul

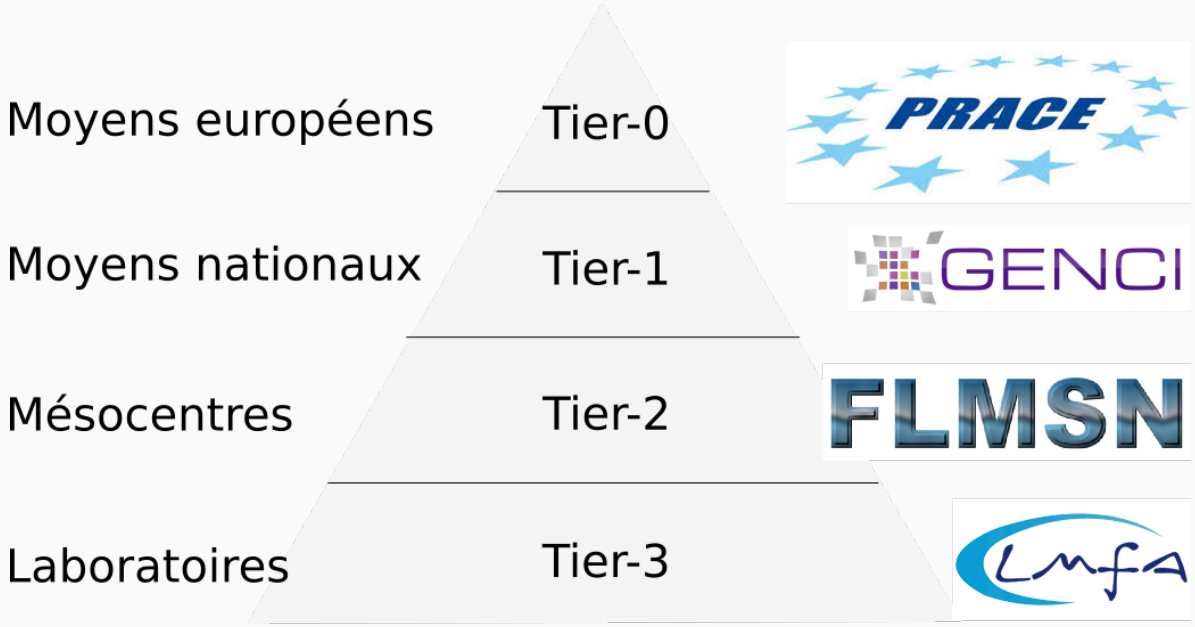
- modélisation numérique
- traitement de données



Contexte

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

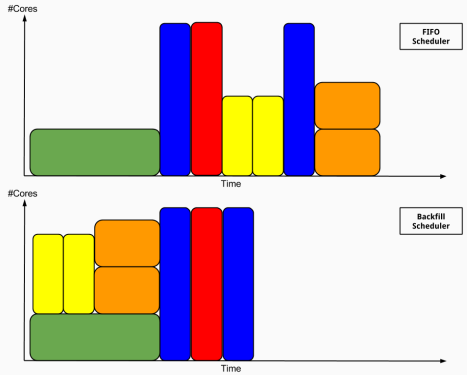
des cœurs, de la mémoire, des GPU, ...

Principe général du fonctionnement des calculateurs

Des utilisateurs

Des ressources de calcul

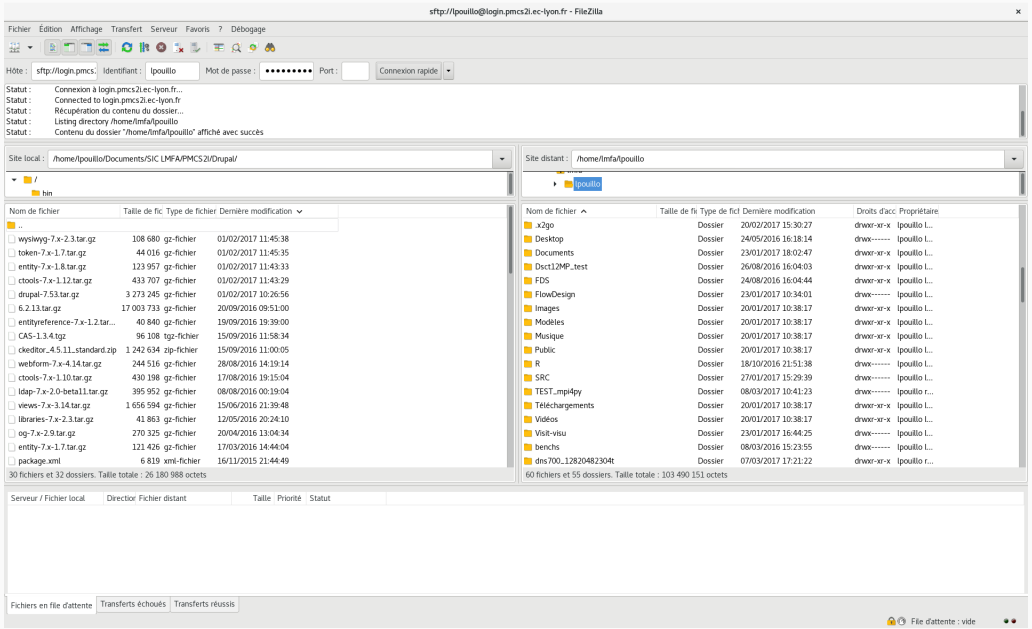
Un ordonnanceur de job



Optimise l'utilisation des resources

Comment y accède-t-on ?

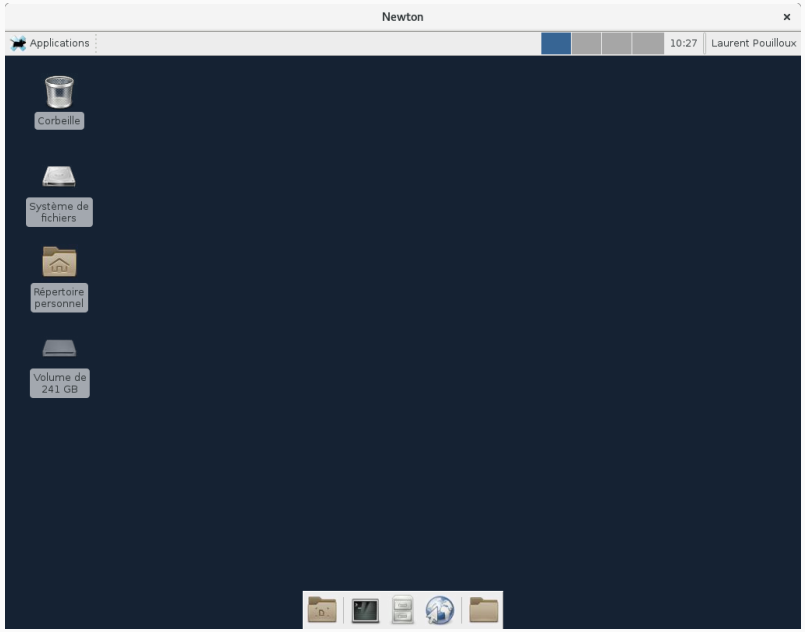
- connexion SFTP



FileZilla

Comment y accède-t-on ?

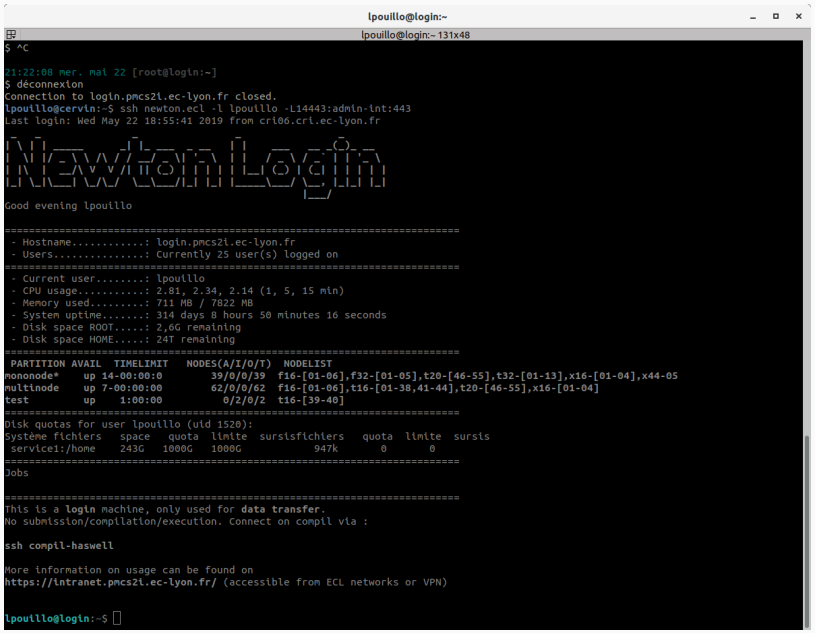
- connexion SFTP
- bureau graphique



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
- nécessité de travailler sur la frontale

Utilisation de moteurs d'expériences

Pourquoi faire ?

- tests de scaling
- diagramme de régime pour un processus physique
- traitement distribué de données

Utilisation de moteurs d'expériences

Pourquoi faire ?

- tests de scaling
- diagramme de régime pour un processus physique
- 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 ?

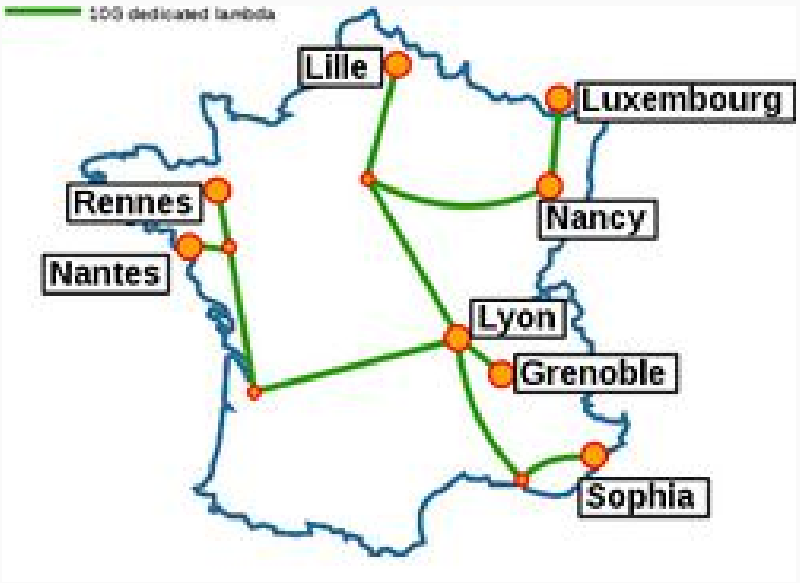
- tests de scaling
- diagramme de régime pour un processus physique
- traitement distribué de données

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**, de réaliser des tâches d'administration et de scripter des expériences de calcul numérique

La solution Execo

API Python permettant de contrôler des **processus unix locaux ou distants**, de réaliser des tâches d'administration 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

Gérer des processus avec 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 moteurs d'expériences

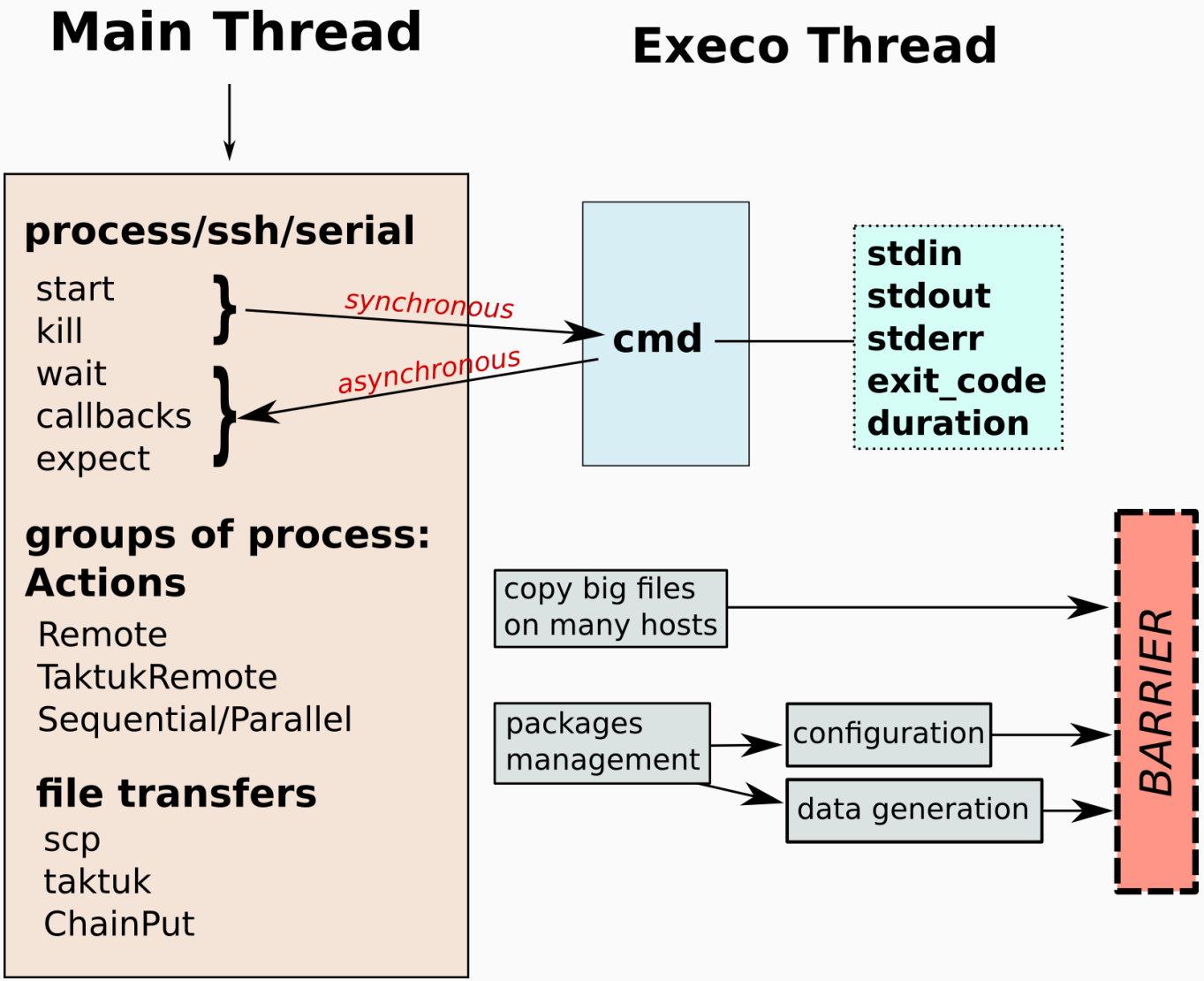
Installation d'Execo

- 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

`http://execo.gforge.inria.fr/`

Vue générale d'Execo



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

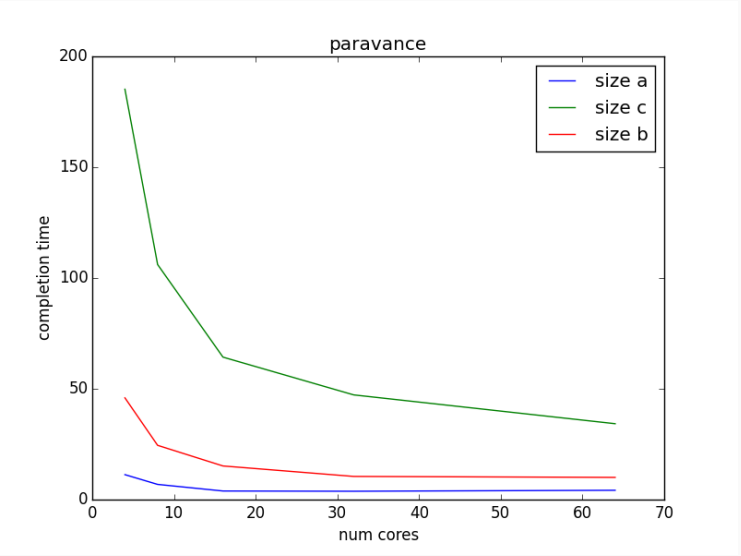
- Héritage des méthodes de **Process**
- Cible : un hôte distant (hostname, user, port, keyfile)

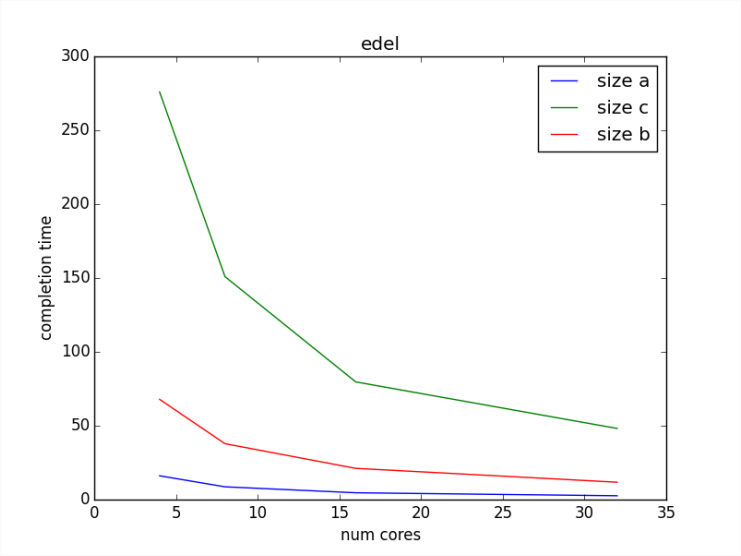
```
In [1]: from execo import SshProcess
In [2]: test = SshProcess('ls /tmp', 'newton').run()
In [3]: print test.stdout
systemd-private-5cc33162c1a94b699ce99258a35a59ad-munin-node.serv
systemd-private-5cc33162c1a94b699ce99258a35a59ad-ntpd.service-5G
tmux-5544
In [4]: print test.host.address
newton
```

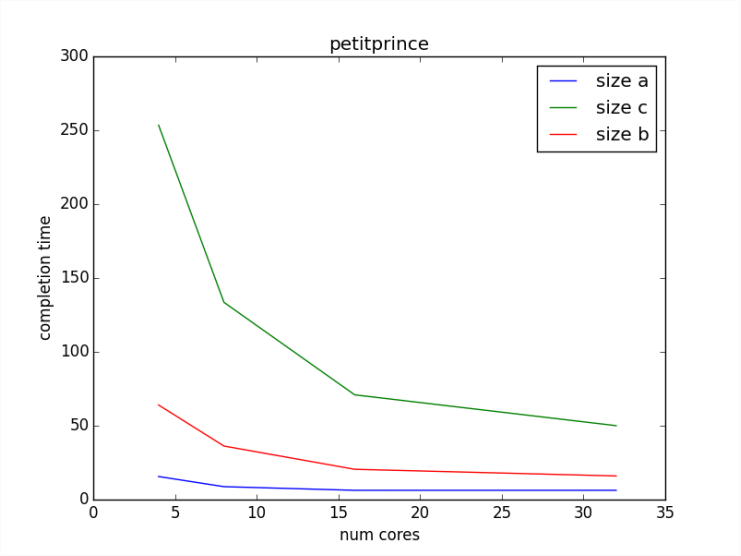
Remote $\Rightarrow$ exécution parallèle sur différents hôtes

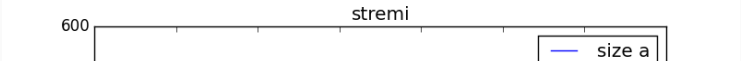
Transfert de fichiers

- **Put**, envoyer les fichiers sur une ou plusieurs machines



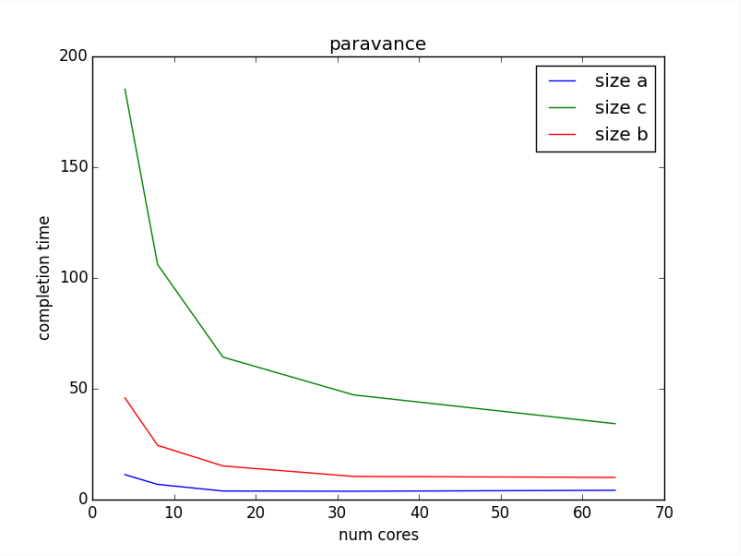


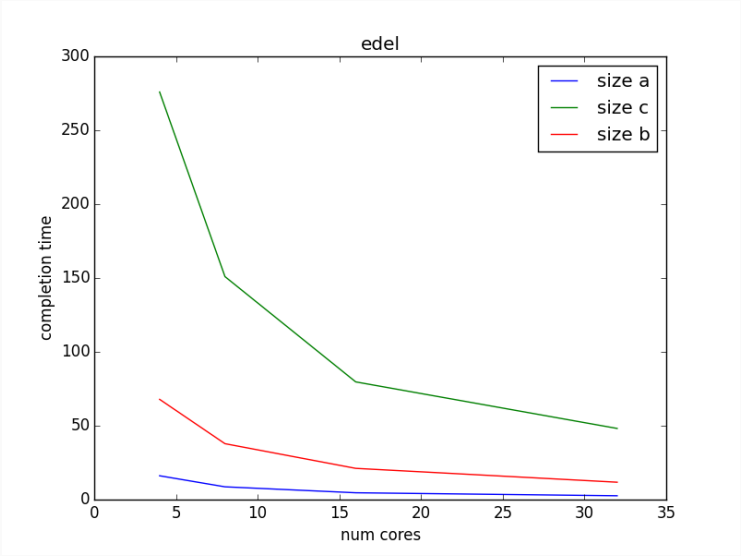


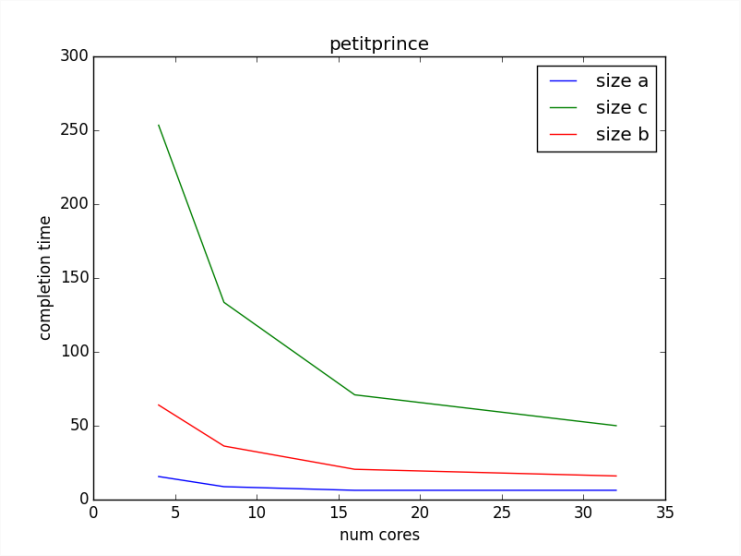


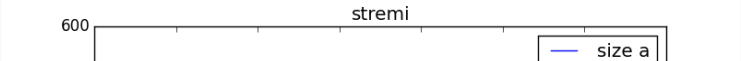
Transfert de fichiers

- **Put**, envoyer les fichiers sur une ou plusieurs machines



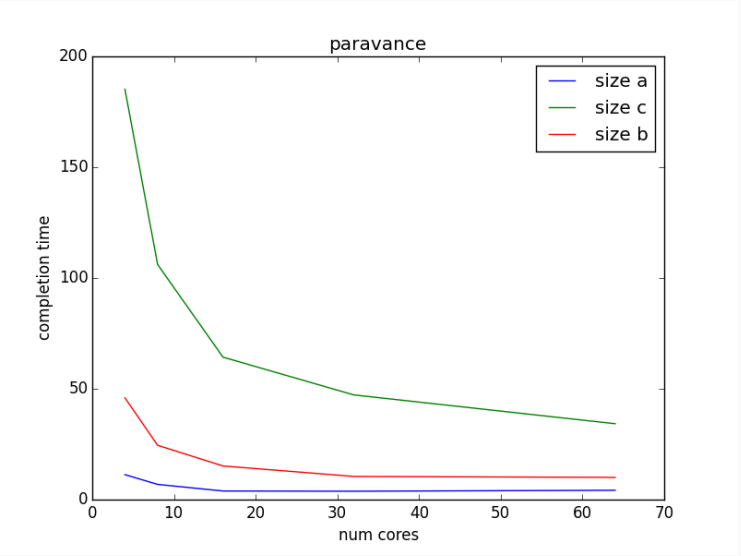


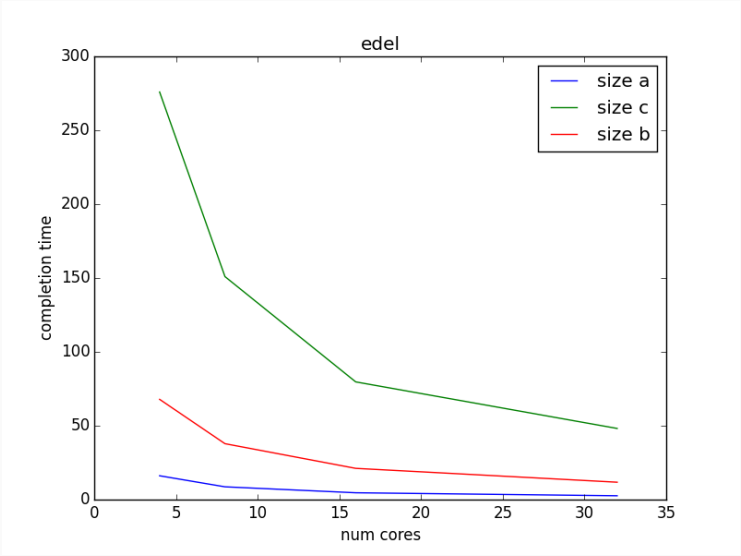


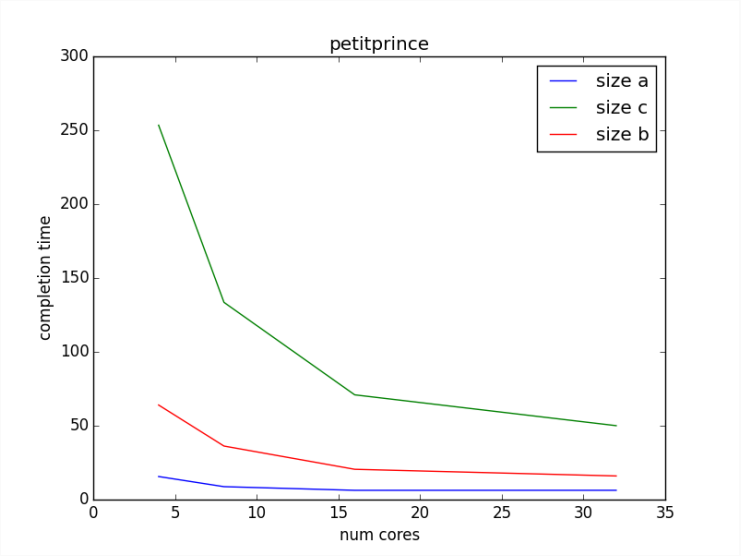


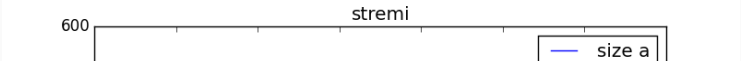
Transfert de fichiers

- **Put**, envoyer les fichiers sur une ou plusieurs machines



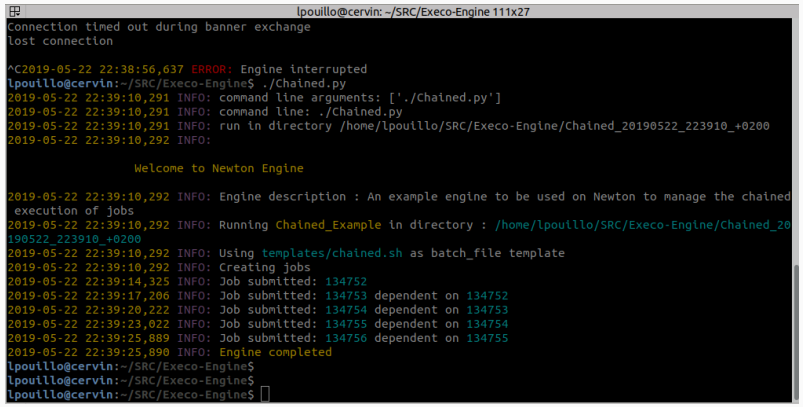






Gestion des logs

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



Deux exemples personnels

- création des comptes sur le calculateur depuis la machine d'admin
  - récupération des informations de l'utilisateur
  - création du compte LDAP
  - mise en place des quotas $\Rightarrow$ SshProcess sur le serveur de fichier
  - création de l'account SLURM $\Rightarrow$ SshProcess sur la machine SLURM
  - execution des tests $\Rightarrow$ SshProcess sur la machine de compilation
  - copie clé SSH $\Rightarrow$ Put dans le home utilisateur
  - envoi des emails
- Réalisation en parallèle des opérations sur iDRAC
- Extinction des serveurs LMFA en cas de coupure de fluides

Campagnes automatisées avec 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

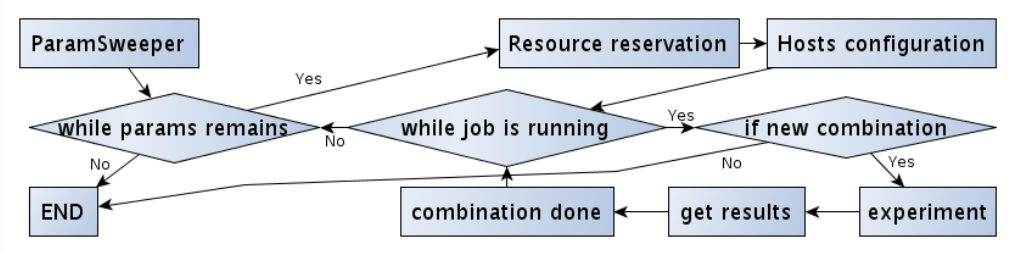
ParamSweeper : objet permettant le balayage simplifié des paramètres

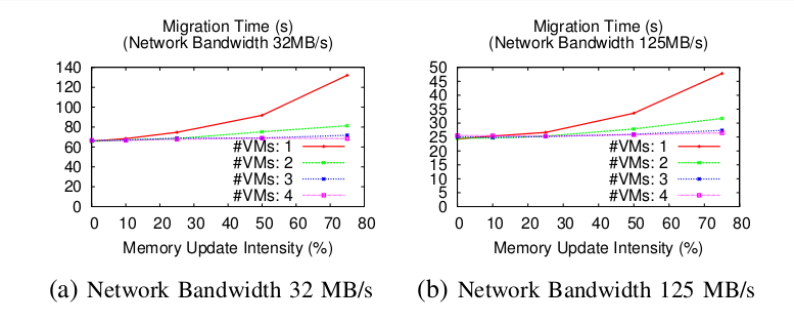
```
In [23]: sweeper = ParamSweeper('test', sweeps)
In [24]: ls test
done  inprogress
```

- 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)`

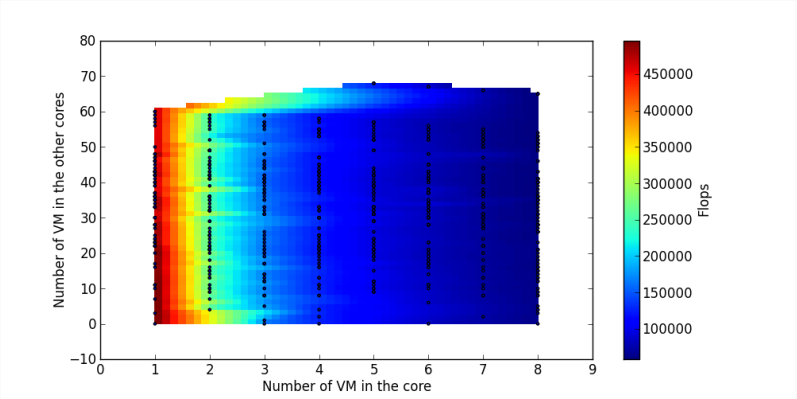
Avantages majeurs : simplification de la gestion du cycle de vie de l'expérience, possibilité de reprise, accès en parallèle

Migrations automatiséees et reproductibles de machine virtuelles





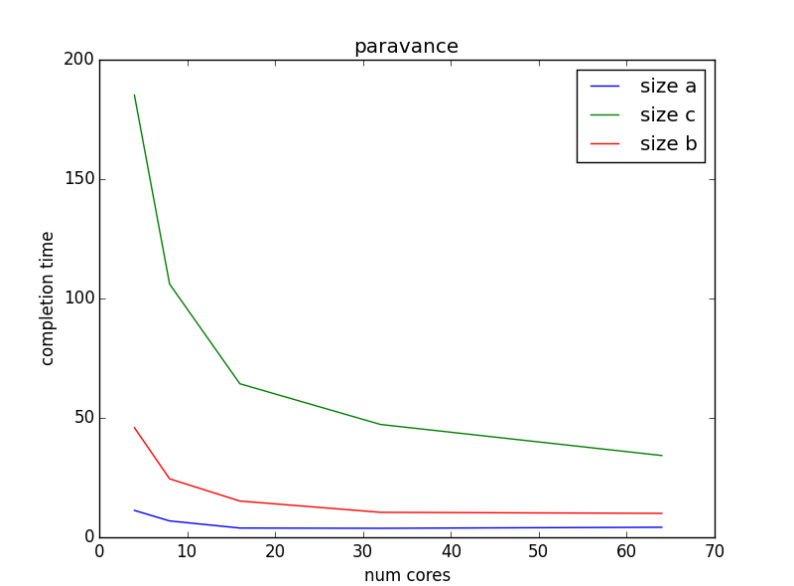
(a) Network Bandwidth 32 MB/s (b) Network Bandwidth 125 MB/s

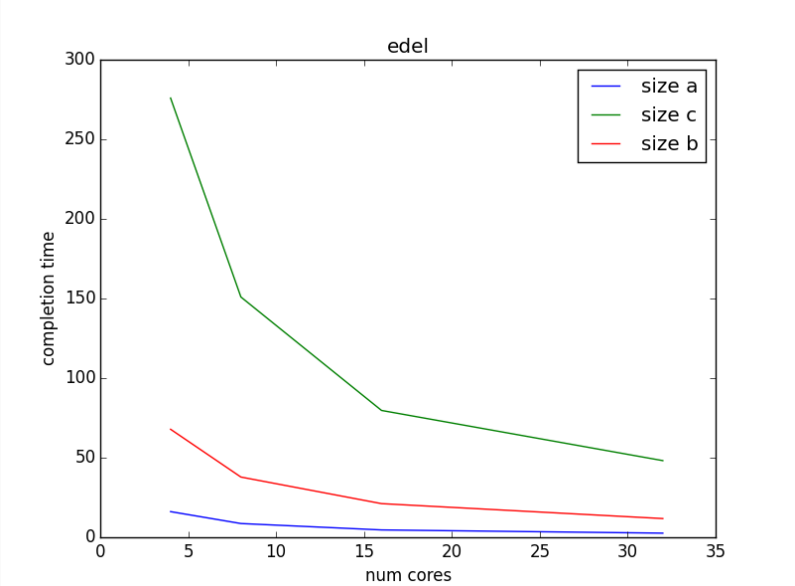


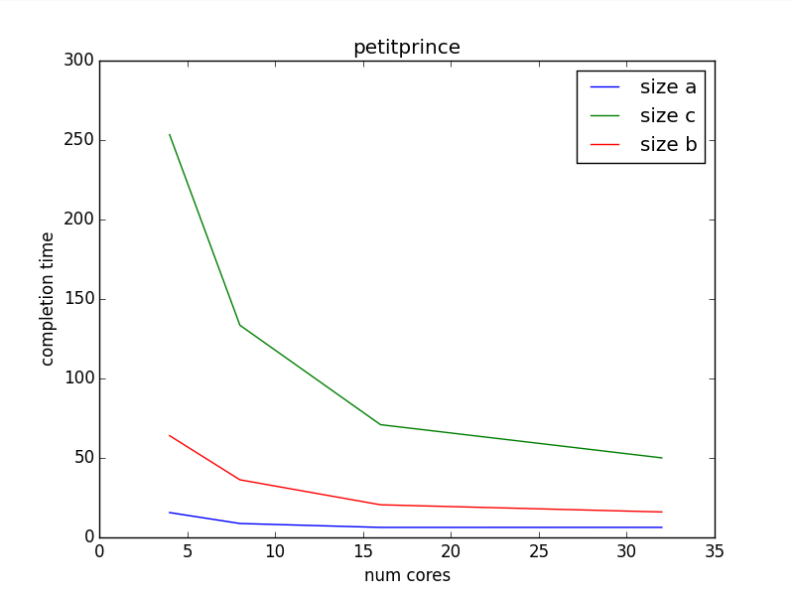
Using the EXECO toolbox to perform automatic and reproducible cloud experiments Matthieu Imbert, Laurent Pouilloux, Jonathan Rouzaud-Cornabas, Adrien Lebre, Takahiro Hirofuchi

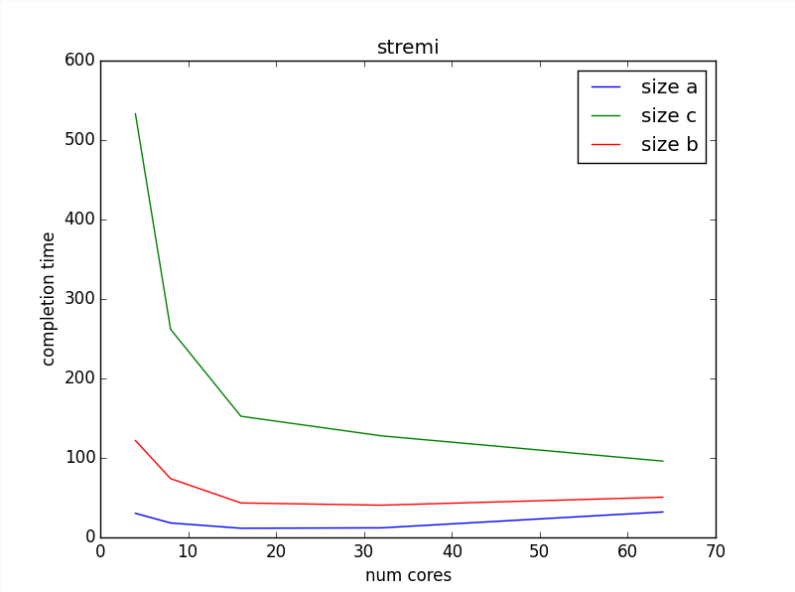
`https://hal.inria.fr/hal-00861882`

Évaluation de la performance MPI des clusters Grid'5000









https://www.grid5000.fr/w/Execo_Practical_Session

Scalabilité des versions de Python

- définition de l'espace de paramètres

```
parameters = {'n_cores': igeom(1, 16, 5),
              'Python': ['system',
                         '2.7.10_intel_2015a',
                         '2.7.11_foss_2016a',
                         '3.4.3_intel_2015a',
                         '3.6.3_foss_2017b']}
```

Scalabilité des versions de Python

- définition de l'espace de paramètres
- définition d'un template de soumission

```
 ..
#SBATCH --ntasks=$n_cores
# Time allocation
#SBATCH -t 1:00:00

$load_module
env | grep -i SLURM |sort
# Execution
python bench.py
```

Scalabilité des versions de Python

- définition de l'espace de paramètres
- définition d'un template de soumission
- soumission des jobs

```
2019-05-23 10:35:28,626 INFO: Starting parametric exploration
2019-05-23 10:35:28,630 INFO: Treating 21/25 in
n_cores-8-python-2710_intel_2015a
2019-05-23 10:35:31,263 INFO: Job 134814 submitted
2019-05-23 10:35:37,702 INFO: Job started
2019-05-23 10:36:32,832 INFO: Files retrieved in
DemoAramis/n_cores-8-python-2710_intel_2015a
2019-05-23 10:36:32,843 INFO: Treating 22/25 in
n_cores-16-python-343-intel-2015a
```

Scalabilité des versions de Python

- définition de l'espace de paramètres
- définition d'un template de soumission
- soumission des jobs
- récupération et analyse des données

```
n_cores-8-python-363-foss-2017b
   bench.py
   job_134792.err
   job_134792.out
   python_scaling.sh
n_cores-8-python-system
   bench.py
   job_134798.err
   job_134798.out
   python_scaling.sh
stdout+stderr
```

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