|  |
| --- |
| Guide Technique |
| Projet de recherche : EPOC |
| **Placement de machines virtuelles dans un centre de données alimenté par des énergies renouvelables.** |

[Choisir la date]

Sommaire

# Introduction

Développé par plusieurs universités, ce projet a pour but d’adapter la charge de travail (et donc la consommation électrique) des serveurs et ressources informatiques en fonction de la disponibilité en énergies vertes.

Il s’agit donc d’une étude d’impact de la consommation électrique des serveurs informatiques.

On distinguera 2 types de job (tâche)

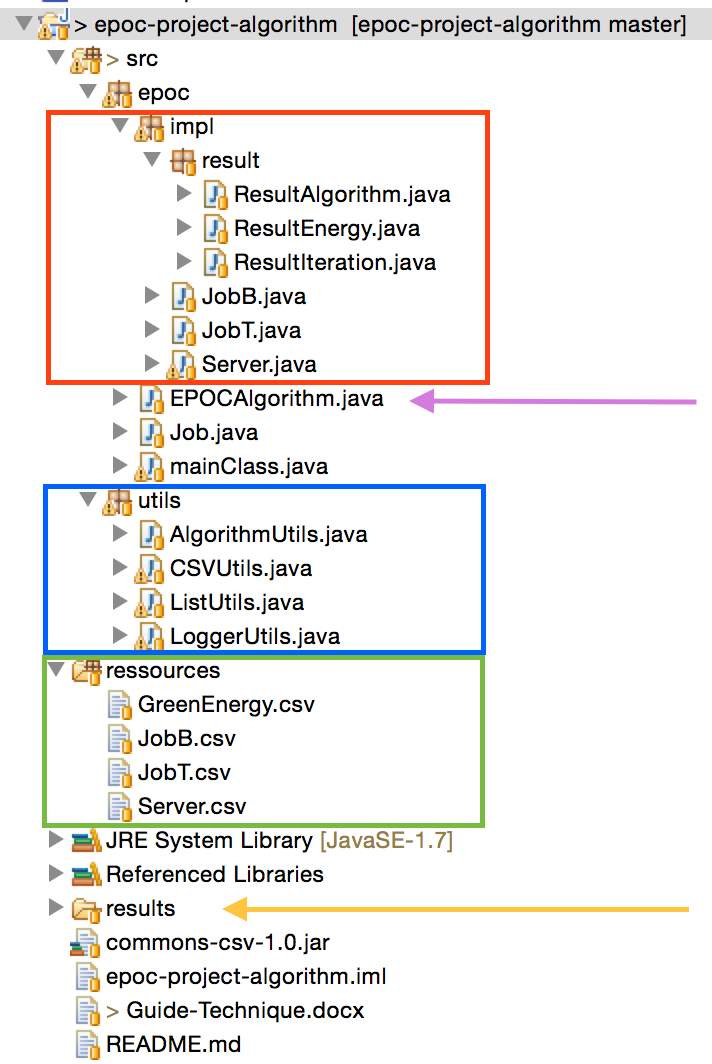
* serveur Web (job T) : tournent tous les jours, nous n’avons pas la main sur leur exécution
* batch (job B) : exécutés sans contraintes calendaires

L’exécution des jobs a besoin de X ressources énergétiques par heures. Toutefois, la charge de travail et la disponibilité en énergies vertes ne sont pas régulières dans le temps. Il faut trouver un moyen d’alimenter les serveurs en utilisant au maximum les énergies renouvelables afin de minimiser notre empreinte énergétique.

Notre piste d’action : Agir sur la charge de travail afin d’adapter la consommation à la disponibilité en énergies vertes.

Pour cela, il faudra optimiser la répartition des tâches de type B (batch) dans le temps afin de minimiser la consommation d’énergie EDF et quantifier l’énergie verte gaspillée (i.e. non utilisée).

# Architecture du projet



Package impl : contient toutes les entités/objets

Classe EPOCAlgorithm : classe principale, implémentation de l’algorithme 3, c’est-à-dire le placement des jobs sur les différents serveurs.

Package utils : ensemble des outils utilisés pour la manipulation des données sous forme de liste, la lecture de CSV etc.

Package ressources : les fichiers csv utilisés en entrée de l’algorithme (jobs à placer, disponibilité énergétique et liste de serveurs, cf. II – Format des données)

Dossier results : contient les logs et autres résultats suite à l’exécution de l’algorithme.

# Format des données

## Fichiers en entrée

### JobB.csv / JobT.csv

Chaque ligne correspond à un job, sous la forme :

ID, itération de départ, liste de charges

Exemple :

302,0,8,9,8,9,8,9,10

303,3,8,9,8,9,8,9,10

Contrainte : Il faut veiller à ce que chaque ID utilisé soit unique.

### GreenEnergy.csv

Contient simplement une liste de disponibilité énergétique, une valeur par itération de l‘algorithme.

Exemple :

450,375,364,467,413,367,375,294,463,345,354

Contrainte : Aucune

### Servers.csv

Chaque ligne correspond à un job, sous la forme :

ID, consommation électrique par charge CPU (Watts / % CPU)

Exemple :

100,1

101,1

Contrainte : Unicité des ID (y compris vis-à-vis des ID de jobs)

## Fichiers de sortie

On définit un format d’affichage par défaut pour les jobs permettant de connaître leur état.

Ce format sera utilisé pour l’ensemble des fichiers de sortie. Ainsi, on pourra connaître l’avancement de l’exécution d’un job et sa consommation à chaque instant de l’algorithme. Ces informations seront affichées suivant ce format :

JobX :ID[iteration, consommation]

### Server-xxx.txt

Fichier de log propre au serveur d’ID xxx. On y retrouve son état à chaque itération sous la forme :

Server ID [liste de jobs] = charge totale

Exemple :

------- Iteration n°2

Server 100 [JobT:207[2:76], JobB:311[0:1]] = 77

### rejects.txt

Contient la liste des jobs qui n’ont pu être placé sur un serveur à chaque itération

Exemple :

------- Iteration n°0

[JobB:301[0:7], JobB:305[0:4]]

### logs.txt

Fichier de log global, il décrit l’exécution de l’algorithme au fil des itérations en regroupant l’ensemble des job à placer (ce qui inclut des nouveaux jobs et les batchs non terminés à l’itération précédente), la liste des serveurs et leurs états, la liste des jobs rejetés, la liste des jobs terminés et le bilan énergétique.

Exemple :

----------------- ResultIteration n°0 -----------------

----- Nouveaux Jobs :[JobT:200[0:52], JobT:201[0:51], JobT:202[0:14], JobT:203[0:48], JobT:204[0:32], JobT:205[0:13], JobT:206[0:23], JobT:207[0:78], JobT:208[0:10], JobT:209[0:23], JobT:213[0:8], JobB:300[0:7], JobB:301[0:7], JobB:302[0:8], JobB:303[0:8], JobB:304[0:4], JobB:305[0:4], JobB:306[0:10], JobB:307[0:8]]

----- Liste des serveurs:

Server 100 [JobT:207[0:78], JobT:202[0:14], JobT:213[0:8]] = 100

Server 101 [JobT:200[0:52], JobT:203[0:48]] = 100

Server 102 [JobT:201[0:51], JobT:204[0:32], JobT:205[0:13], JobB:304[0:4]] = 100

Server 103 [JobT:206[0:23], JobT:209[0:23], JobT:208[0:10], JobB:306[0:10], JobB:302[0:8], JobB:303[0:8], JobB:307[0:8], JobB:300[0:7]] = 97

----- Liste des rejets: [JobB:301[0:7], JobB:305[0:4]]

----- Jobs Terminés: []

----- Energie utilisée : Used(397.0) - Green energy(450.0) - WastedEnergy(53.0)

----------------------------------------------------------------

# Les algorithmes

## Placement des web jobs

Trie la liste des web jobs par ordre descendant (selon leur consommation) pour l’itération en cours, puis les place uns à uns sur le premier serveur pouvant supporter cette consommation.

## Placement des batchs

Fonctionnement similaire mais dans ce cas, on trouve sur les jobs de type batch.

## Algorithme principal

Cet algorithme est le pilier de l’application. Au début de chaque itération, il retire les batchs de chaque serveur. Il déplace ensuite les jobs web déjà présents sur les serveurs afin d’optimiser leur répartition. On cherche ici à utiliser le moins de serveurs possible, il s’agit d’une liberté que nous avons prise afin d’optimiser la solution proposée qui ne se contentait que de déplacer les jobs faisant dépasser la capacité du serveur.

Il se charge alors d’appeler les deux premiers algorithmes afin de placer les nouveaux jobs web et l’ensemble des jobs de type batch (ce qui inclus ceux non terminés à l’itération précédente et les nouveaux arrivants).