Éco-conception des services numériques

BL.RESEARCH

En JAVA, on fait comment ?

De quoi s'agit-il ?

IDENTIFIER LES HOTSPOTS

L'écoconception des services numériques est une **approche globale** d'optimisation du **code** et de l'**architecture**.

Dans quel but ?

PROPOSER DES CHANGEMENTS D'IMPLÉMENTATIONS

Le numérique est la cause de **4% des émissions mondiales de gaz à effet de serre** (GES).
C'est plus que l'aviation civile !

Comment y prendre part ?

RÉDUIRE L'MPACT DU SERVICE

Optimiser du **code JAVA** permet de limiter l'impact du numérique (**énergie** et **puissance** nécessaires).

Limiter les ressources nécessaires à l'exécution des services permet aussi de limiter le **renouvellement des terminaux** induits par leur utilisation. En améliorant le **code JAVA**, on réduit **l'énergie et la puissance consommées** par les équipements, tout en améliorant la **performance des services**. Les outils de mesure permettent d'identifier les classes et les implémentations les plus **consommatrices** pour mieux les éviter.

Utiliser le moins de librairies possibles

- L'**empilement** des technologies de façon générale entraîne une **surconsommation**.
- Importer des librairies inutiles impacte directement la **consommation en énergie** du programme.

Les classes consomment différemment

- Il est intéressant de monitorer les activités du **CPU** et de la **mémoire** pour choisir les meilleurs outils de développement.

Comparaison des implémentations List

- Les **TreeList** consomment plus que de simples **ArrayList**.
- Idem pour les **Map** et les **Set**, dans lesquels les "Tree" sont encore plus consommateurs.

Gestion des buffers

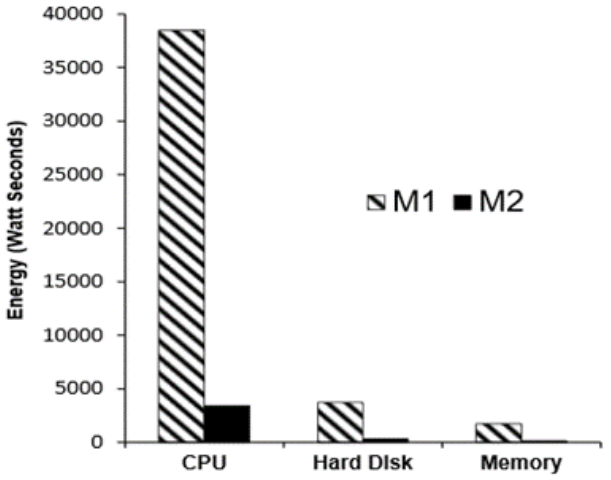
Une bonne gestion de la taille des buffers peut réduire jusqu'à **10 fois** l'énergie nécessaire.
Comparaison de deux méthodes de lecture d'un fichier :

- M1 avec **FileInputString**

```
FileInputStream fis = new FileInputStream(fileName);
int b,cnt = 0;
while ((b = fis.read()) != -1)
{
        if (b == '\n')
                cnt++;
}
fis.close();
```

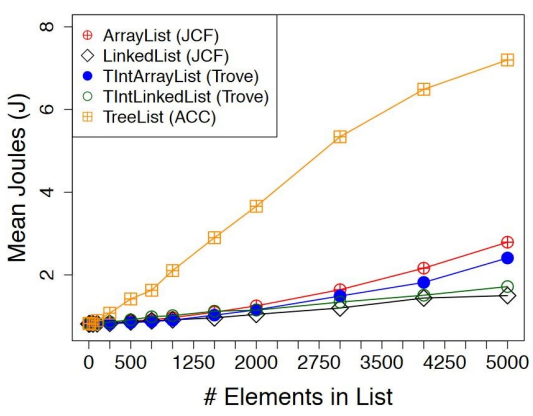
- M2 avec **BufferedInputStream**

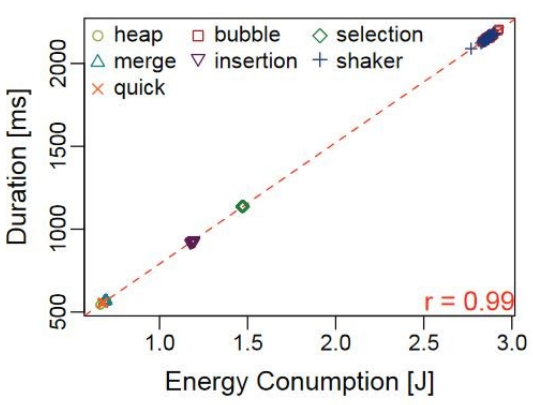
```
FileInputStream fis = new FileInputStream(fileName);
BufferedInputStream bis = new BufferedInputStream(fis);
int b,cnt = 0;
while ((b = bis.read()) != -1)
{
        if (b == '\n')
                cnt++;
}
fis.close();
```



La figure de gauche montre les différences de consommation selon les **types de listes** utilisées.

La figure de droite montre la corrélation **temps d'exécution** / **énergie consommée** pour des algorithmes de tri. Les opérations rapides sont moins consommatrices que des opérations comme les tris à bulle.





Pour aller + loin, rendez-vous sur https://tinyurl.com/BLGreenIT

Berger Levrault