impacts énergétiques

Groupe 19:

Salaheddine ASLOUNE

Wen LUO

Anass BOUBKRI

El Mehdi KHADFY

Amine BELFKIRA

Sommaire:

I. Consommation énergétique de votre projet:

- Afin de mesurer les impacts énergétiques liés à notre surconsommation du numérique, nous utilisons la calculatrice INR(Institut du Numérique Responsable), créée par Decathlon pour mesurer son impact numérique professionnel. Les calculs ont pour unité de mesure: Kg d'équivalent en CO_2. Notre équipe de projet est composée de 5 personnes chacun travaillant sur son pc personnel. Ainsi, notre équipement en 1 mois a eu un impact numérique de 25,87. Notre projet a un stockage d'environ 38 mo en cloud, cela a un impact de 4.5 par pc soit 22.5 au totale. Le trajet de travail des membres de l'équipe a un impact de 6,629. Au total, l'impact énergétique de l'équipe pendant tout un mois peut être estimé à 55 (KG EQ CO_2).
- Une consommation de 55 est assez importante, puisque un eucalyptus de 8 mètres de haut ne peut stocker en une année qu'environ 50-60 kg de CO_2.

II. Choix de génération de code:

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

Tableau des valeurs normalisées de l'énergie consommée, du temps d'exécution, et de la mémoire occupée pour chaque langage. source i/

- Comme on peut le voir dans le tableau ci-dessus, les consommations énergétiques diffèrent selon les langages de programmation. C'est pourquoi nous avons préféré écrire des scripts en langage shell plutôt qu'en python, puisque ce dernier se situe au bas de la liste, et donc c'est un très grand consommateur d'énergie. Aussi, nous avons cherché à nous rassembler tous les jours afin d'éviter les réunions numériques et de travailler en groupe avec l'intention de résoudre les problèmes rapidement et donc de réduire la durée d'utilisation de nos équipements.

III. Choix de processus de validation:

- Pour nous assurer que le code remplit son objectif, nous avons utilisé plusieurs tests pour valider nos modifications. Nous utilisons également plusieurs scripts pour lancer automatiquement l'exécution de tous les tests et vérifier qu'ils fonctionnent correctement, cela économise considérablement le temps de consommation numérique. Nous avons cessé de répliquer les mêmes formes de tests qui ne changeront pas la couverture du code, mais nous écrivons plutôt différents tests qui sont complets et couvrent chacun une quantité maximale de code.

Bibliographie:

Source : https://programmation.developpez.com/actu/296868/