**Bilan comparatif des protocoles d'évaluation**

Le bilan comparatif des protocoles d’évaluation nous permet de faire une analyse des paramètres et des valeurs utilisés pour évaluer chaque protocole. L’objectif final de cette analyse est de définir un protocole expérimental qui nous servira de base d’analyse.

# Rappel sur les Heuristiques d’ordonnancement des tâches

L'ordonnancement est un processus qui consiste à partitionner les applications en tâches et les affecter à des ressources pour traitement. Ces applications sont modelisées par un graphe acyclique . Où représente l’ensemble tâches, et représente les liens entre les tâches.

# Choix des heuristiques

Dans le cadre de cette étude, nous allons nous allons travailler avec les 5 heuristiques suivants : HEFT,PEFT,PETS,DLS,CPOP.

En effet la littérature présente l’heuristique PEFT comme celle qui offre les meilleures performances. Selon l’article [[1]](#footnote-1) PEFT est la première heuristique dont les performances surclassent celle de HEFT. Il est question dans notre étude d’examiner les bases qui ont servi d’analyse et proposer un protocole expérimentale qui nous permettra d’approuver ou non cette conclusion.

# Analyse des paramètres d’évaluation

Nous distinguons deux grandes catégories de paramètres : les paramètres de genération des graphes et les paramètres de comparaison.

## Paramètre de génération des graphes

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | PEFT | HEFT | CPOP | PETS | DLS |
|  | Taux de communication . Si la valeur est faible alors il s’agit d’un graphe offrant une communication intensive entre les tâches | √ | √ | √ | √ | √ |
|  | Nombre de tâches | √ | √ | √ | √ | √ |
|  | Nombre de processeurs | √ | √ | √ | √ | √ |
|  | Forme du graphe du graphe si >1.0 graphe dense, si <1.0 graphe avec un dégrée de parallélisme faible | √ | √ | √ |  |  |
|  | Dégrée sortant d’un noeud |  | √ | √ |  |  |
|  | Facteur d’hétérogenéité | √ | √ | √ |  |  |
|  |  | √ |  |  |  |  |
|  |  | √ |  |  |  |  |
|  |  | √ |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## Paramètres de comparaison des graphes

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | PEFT | HEFT | CPOP | PETS | DLS |
|  |  | √ | √ | √ |  |  |
|  |  | √ |  |  |  |  |
|  |  | √ |  |  |  |  |
|  |  | √ | √ | √ |  |  |
|  |  | √ |  |  |  |  |
|  |  | √ |  |  |  |  |
|  |  | √ |  |  |  |  |
|  |  |  | √ | √ |  |  |
|  |  |  | √ | √ |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

D’un algorithme à un autre les métriques de comparaison peuvent varier. Les paramètres les plus significatives pour la comparaison

Génération des graphes aléatoire ???

Signaler comment cest graphes sont générés, est ce qu’il utilisent des biliothèques connus open sources ou des bibliothèques propre.

# Choix des valeurs

## Génération des graphes

Les valeurs utilisées dans les différents algorithmes pour générer les graphes sont les suivantes :

**PEFT** : n = [10; 20; 30; 40; 50; 60; 70; 80; 90; 100; 200 300*;* 400*;* 500], CCR =[ 0:1; 0:5; 0:8; 1; 2; 5; 10]

β [0:1; 0:2; 0:5; 1; 2], Processors [2; 4; 8; 16; 32]

**HEFT** : n = [ 20; 40; 60; 80; 100], CCR =[ 0:1; 0:5; 1.0; 5.0; 10.0], β [0:1; 0:25; 0:5; 0.75;1]

**CPOP :** n = [ 20; 40; 60; 80; 100], CCR =[ 0:1; 0:5;; 1.0; 5.0; 10.0], β [0:1; 0:25; 0:5; 0.75;1]

**PETS** :

**DLS** :

Nous pouvons constater une forte différence entre le choix des valeurs soumises à la génération des graphes de tâche. L’algorithme PEFT considère des graphes dont le nombre maximal de tâche est de 500 ce qui n’est pas le cas pour HEFT dont les valeurs maximales sont limités à 100. On comprend bien que les bases d’évaluations ne sont pas les mêmes pour les différents articles. Plus le nombre de tâches sera grand plus nous aurons la possibilité de faire une évaluation objective. L’intérêt de ce travail relève de la nécessité de définir un protocole qui servira de de base d’évaluation de ces différents heuristiques.

# PROPOSITION D’UN PROTOCOLE EXPERIMENTAL

## Notre méthodologie de comparaison est basée sur la conception d’un simulateur. Le simulateur nous permettra d’une part de générer les graphes de tâches, implémenter les différents algorithmes dans un langage de programmation, exécuter ces programmes, générer des résultats et ensuite proceder à une analyse des résultats obtenus.

## Génération des DAG

Les DAG seront générés de façon aléatoire grâce à un programme écrit en langage python. Ce programme a été mis à notre disposition par notre encadreur. Il s’agit du résultat d’un trravail réalisé par un autre étudiant. La génration se fait grace à l’utilisation des bibliothèques libres : igraphToNetworkX, igraph,numpy.

La liste des paramètres de génération de graphe sont les suivants :

* length: Length of the graph (number of nodes)
* :param depth: Depth of the graph (number of levels)
* :param filename: Output filename
* :param sdComp: Standard Deviation of computations costs
* :param sdComm: Standard Deviation of communications costs
* :type sdComm: float
* :param CCR: Communications to Computations Ratio
* :param nbproc: Number of processors used when generating the graph

# Description des paramètres

|  |  |
| --- | --- |
| length | Nombre de nœuds du ggraphe |
| depth | Profondeur du graphe. Ce paramètre permet de définir le nombre de niveaux du graphe |
| sdComp | Facteur de déviation. Il s’agit d’un paramètre qui permet de de définir le dégré d’hétérogenéité d’un graphe |
| sdComm | Ce paramètre permet de définir le niveau d’hétérogenéité des coûts de communication |
| CCR | Ce paramètre permet de définir le taux de communication. Pour une valeur faible, il s’agit d’un graphe dont la communication est intensive, |
| nbproc | Nombre de processeurs |
|  |  |

Grâce à ces paramètres nous seront capable de générer des DAG de différentes topologies, ainsi que des processeurs disposant des vitesses variées. Le coefficient de variation des processeurs nous permettra de définir le niveau d’homogénéité des processeurs.

Comme resultat de la génération aléatoire des graphes, nous obtenons des fichier de type gml. Il s’agit du format de fichier natif à python qui permet de définir les graphes.

# Repoductibilité de la recherche

Pour rendre reproductible notre travail, c’est-à-dire s’assurer que toute personne utilisant les mêmes valeurs que nous sera capable de produire le même résultat, nous utiliserons les techniques de ***seed (graine aléatoire)***.

Selon wikipedia, une graine aléatoire (aussi appelée germe aléatoire) est un nombre utilisé pour l'initialisation d'un [générateur de nombres pseudo-aléatoires](https://fr.wikipedia.org/wiki/G%C3%A9n%C3%A9rateur_de_nombres_pseudo-al%C3%A9atoires).

Toute la suite de nombres aléatoires produits par le générateur découle de façon déterministe de la valeur de la graine.

# Choix des valeurs

La littérature présente le PEFT comme étant l’heuristique qui offre la meilleure performance. Nous nous proposons donc dans le cadre de ce travail de maintenir les mêmes valeurs de génération des graphes qui ont été prise compte dans l’article PEFT. Ces valeurs sont :

* n = [10; 20; 30; 40; 50; 60; 70; 80; 90; 100; 200 300*;* 400*;* 500],
* CCR =[ 0:1; 0:5; 0:8; 1; 2; 5; 10]
* β [0:1; 0:2; 0:5; 1; 2]
* Processors [2; 4; 8; 16; 32]
* α

Nous procéderons à La modification de ces paramètres de manière à de définir des topologies différentes des DAG. La génération des DAG de différente topologie permette de réaliser une analyse pertinente afin d’obtenir des résultats non biaisés qui prennent en compte les différentes configuration des DAGs .

Ces valeurs représentent une évolution linéaire du nombre des nœuds et permettent d’observer si les résultats obtenus en fonction du nombre des nœuds et du nombre de processeurs restent constant ou alors varient. Dans le cas d’une évolution constante nous auront des arguments sur les garanties de la qualité des résultats.

# Programmation du simulateur

Notre simulateur sera écrit en langage java. Comme donnée d’entrée nous aurons les graphes générés en python sur le format gml. Notre simulateur intègrera un un parseur permettant d’analyser les fichier gml afin de générer des DAG.

Un DAG n’est rien d’autre qu’un graphe orienté sans cycle.

**Métriques de mesure des performances**

La performance d’un Heuristique est évaluée en calculant le makespan qui mesure le temps d’exécution de la tâche de sortie d’une application dans un environnement de calcul hétérogène. Le makesapn n’est pas suffisant pour analyser les perfomances de plusieurs heuristiques. Il faut donc faire appel à de nouveau paramètre ou métrique d’analyse. L’analyse s’opère en générant les graphes de différentes topologies. Il est donc question d’évaluer la moyenne des performances obtenues par chaque heuristique pour chaque topologie

Le critère de base sur lesquels nous allons nous appuyer dans notre étude sont les suivants :

* SLR
* makespan
* Nombre d’occurences du meilleur résultat

# Analyse des performances

Les donnée issues du simulateur sont les suivantes : makespam, temps d’exéxution de chaque algorithme. Ces données seront compilées dans un fichier sous le format csvsuivant :

Nom de l’algorithme ;valeur du makespan ;temps d’exécution ; SLR

Un analyseur de performance sera écrit en langage python. Ce programme prendra comme données d’entrée les fichier csv et en sortie nous obtiendrons des graphes qui nous. Les types de graphes attendus sont les suivants :

1. Performance-effective and low-complexity Task Scheduling for Heterogeneous computing By Haluk Topcuoglu [↑](#footnote-ref-1)