IV .1°) Présentation de la première solution

Deux solutions prometteuses d'allocation de ressource sont présentées et comparées avec une autre solution pour la validation de leurs performances. La première solution est basée sur l'énumération d'algorithme et le second sur le Clustering-heuristic. L'objectif est de résoudre le problème de minimiser le temps de réponse moyen de services par l'allocation des ressources de bord optimale pour les applications.

IV .1.1°) Architecture étudiée

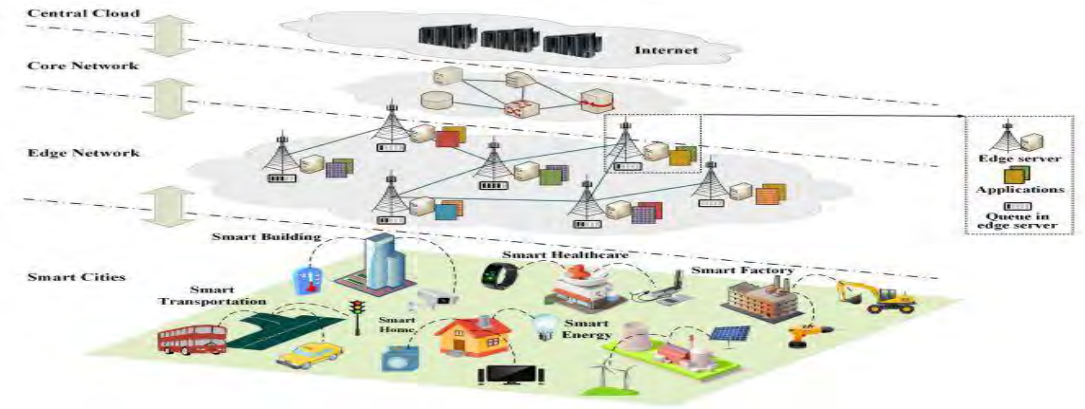

Figure 26 L'architecture de réseau pour les villes intelligentes basées sur l'internet des objets[]

Fonctionnement de l'architecture et les solutions

Nous considérons un scénario où les serveurs allouent des ressources à plusieurs applications pour prendre en charge les appareils IoT exécutant divers services de ville intelligente. Un nombre croissant de villes intelligentes basées sur l'IoT, les services ont des contraintes de latence strictes, par conséquent, le temps de réponse sera critique pour l'utilisation de ces services. Nous présentons plusieurs analyses sur le service moyen de minimisation du temps de réponse en allouant de manière optimale les ressources de périphérie pour plusieurs applications.

De la Figure 26, il peut être observé clairement que le temps de réponse du service se compose principalement de trois parties: le délai de transmission des données dans les réseaux périphériques, le délai de traitement des services dans les réseaux périphériques et le retard du réseau vers le nuage central pour les demandes excédentaires.

Lorsque toutes les demandes de service des appareils IoT arrivent aux serveurs périphériques les plus proches hébergeant les applications correspondantes, les serveurs affectent les ressources informatiques à chaque requête. La capacité limitée de chacun des serveurs Edge pour traiter les demandes de service est également pris en considération. Un serveur Edge comprend un nombre de machines physiques interconnectées pour traiter les demandes de service entrantes, nous considérons chaque serveur de périphérie comme une unité abstraite à gérer les demandes attribuées. Par conséquent, il convient de modéliser le traitement des demandes de service dans chaque serveur Edge comme modèle de mise en file

d'attente. Lorsque les serveurs de périphérie sont surchargés, le service de mode d'allocation passera au Cloud informatique traditionnel. Par la suite, les demandes de services excédentaires sera acheminé vers le Cloud central via le noyau réseau et Internet. En se basant sur les informations initiales des ressources (vCPU, Stockage et RAM) demandés par les applications au serveur en périphérique.

Ci-dessous est mentionné le fonctionnement des deux schémas proposés pour résoudre le problème SRTM dans les villes intelligentes basées sur l'IoT.

- **Énumération-Based Optimal Edge Resource Allocation Algorithm (Ressource Edge optimale basée sur l'énumération) :**

Une solution de dénombrement optimale pour SRTM principalement contient trois étapes. Tout d'abord, il énumère tous les possibles combinaisons d'allocation de ressources de bord pour plusieurs applications en utilisant les algorithmes standard d'ordonnancement. Ensuite, il calcule le service moyen temps de réponse de chaque cas d'allocation de ressources. Enfin, une combinaison optimale d'allocation de ressources pour plusieurs applications peut facilement être trouvée en comparant les résultats calculés. Pour chaque cas d'allocation de ressources, nous considérons l'affectation de demandes de service de périphériques IoT à serveurs de bord de manière équilibrée. Cette procédure délivre ces requêtes au serveur Edge le plus proche héberger les applications corrélatives et posséder des ressources inoccupées. En même temps, ça prend également en compte l'équilibrage de charge entre les serveurs de bord réalisables en affectant les demandes aux serveurs périphériques disponibles à leur tour.

- **Clustering-Based Heuristic Edge Resource (Ressource Edge heuristique basée sur le clustering)**

Ressource Edge heuristique basée sur le clustering (CHERA) est un algorithme d'allocation ressource permettant de résoudre le problème de la SRTM à une échelle réaliste dans un délai raisonnable.

Tout d'abord, CHERA suppose que tous les serveurs de périphérie sont sans charge et sont des candidats pour allouer des ressources pour chaque application. CHERA sélectionne le serveur de périphérie pouvant obtenir un service moyen minimal de temps de réponse dans les situations de réseau de périphérie actuelles pour allouer des ressources à l'application.

- **Greedy Edge Resource Allocation (GERA)**

L'algorithme GERA (Edge Resource Allocation) est également présenté pour valider les performances des algorithmes proposés. Il est bien connu que si un serveur Edge reçoit plus de la moitié des demandes globales d'application, la valeur de la solution optimale d'allocation de ressources pour les applications inclura ce serveur Edge. GERA de manière itérative évalue chaque serveur Edge candidat à allouer de ressources pour les applications par le temps moyen de réponse du service à condition que plus de la moitié des demandes d'application sont acheminées vers le serveur de périphérie. Puis, après avoir goulûment sélectionné des serveurs Edge pour allouer des ressources à chaque application avec un temps de réponse moyen du service minimal, GERA peut obtenir une solution presque optimale.

IV .1.3°) Resultats

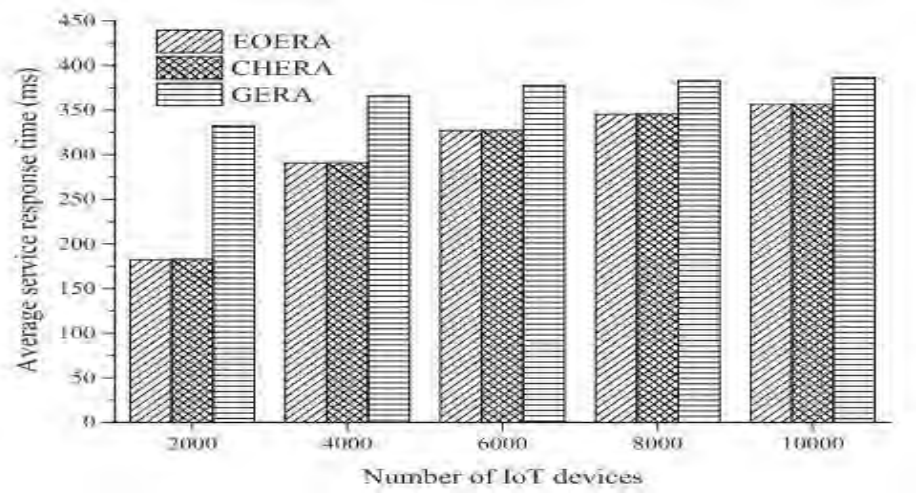

Figure 27 Performance comparisons of EOERA, CHERA, and GERA under different number of IoT-devices.

Dans la Figure27, on peut facilement observer qu'avec le même nombre d'appareils IoT, EOERA peut obtenir les meilleures performances pour minimiser le temps de réponse moyen du service. De plus, les performances CHERA et EOERA sont beaucoup plus proches et sont à la fois mieux que celui de GERA sous différents nombres d'appareils IoT. La figure 25 montre également que la tendance à la croissance du temps de réponse moyen des services ralentit vers le bas avec un nombre croissant d'appareils IoT dans tous régimes. Ceci est principalement dû au fait que l'utilisation des ressources du bord a été saturée, et le service supplémentaire des demandes causées par l'augmentation des appareils IoT être transmis au nuage central, ce qui provoque un délai réseau constant pour chaque application de nos expériences. Il est clairement montré sur la Fig.27 que lorsque les ressources de pointe dominent le traitement des services demandés, CHERA peut toujours obtenir des performances bien meilleures que celles de GERA, et est beaucoup plus proche de la solution optimale obtenue par EOERA. Des conclusions similaires peuvent être obtenues à partir de la Figure3. Comme nous pouvons le voir, les performances de CHERA dans la résolution du problème SRTM sous différents nombre de demandes est beaucoup plus proche de celui de EOERA est bien meilleur que celui de GERA.

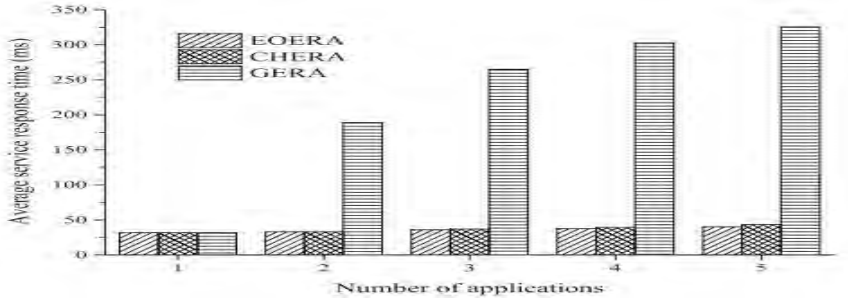

Figure 28 Performance comparisons of EOERA, CHERA, and GERA under different number of applications.

La figure28 montre également qu'avec le nombre croissant d'applications différentes, le service moyen temps de réponse obtenu par EOERA et CHERA grandissent lentement. Chaque serveur Edge a une capacité de service limité, et CHERA remplit de manière itérative le schéma d'allocation de ressources basé sur les serveurs de périphérie candidats pour chaque application, conduisant à des solutions sous-optimales pour les applications. Ainsi, les performances de CHERA est légèrement plus pauvre que celle de l'EOERA avec un nombre d'applications différente. Quand les serveurs de bord n'ont besoin d'allouer des ressources qu'à une seule application, tous les schémas peuvent obtenir les mêmes performances. Cependant, le résultat

de GERA augmente fortement avec le nombre croissant d'applications, ce qui est beaucoup plus élevé que celui de CHERA.

Présentation de la deuxième solution

Cette solution propose une programmation à trois(3) phases afin d'atteindre l'équilibrage de charge et de réduire le temps d'exécution de chaque nœuds dans l'environnement du Cloud. Dans la première phase, la programmation BTO (Best Task Order) détermine l'ordre d'exécution pour chaque demande de tâche. Dans la deuxième phase, la programmation EOLB (Enhanced Opportunistic Load-Balancing) assigne les taches à un gestionnaire de service approprié pour l'attribution du noeud de service. Dans la troisième phase, l'EMM (Enhanced Min-Min) garantit la planification qu'un nœud de service approprié sera affecté à exécuter la tâche dans le temps d'exécution minimum.

IV.2.1°) Architecture étudiée

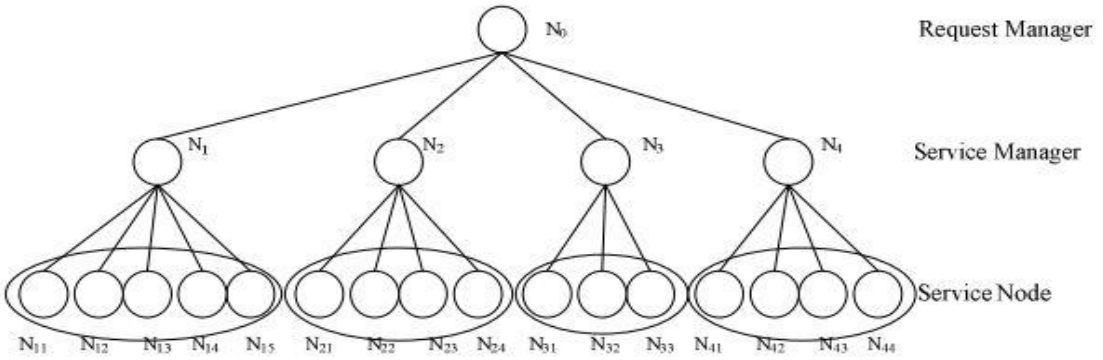

Figure 29 Three-level framework

IV.2.2°) Fonctionnement de l'architecture et les solutions

En raison des propriétés du Cloud computing, la figure27 montre une topologie hiérarchique à trois niveaux. Le troisième niveau est le nœud de service qui sert à exécuter les sous-tâches. Le deuxième niveau est le gestionnaire de services qui sert à diviser la tâche en quelques sous-tâches indépendantes logiques. Le premier niveau est le gestionnaire de requêtes qui permet de confier la tâche à un gestionnaire de services approprié.

Le fonctionnement de ces trois programmations développé dans la solution étudiée est expliqué ci-dessous :

- **BTO scheduling :**

Dans la première phase, le meilleur ordre d'exécution sera organisé par la programmation BTO, après les tâches sont entrées dans l'environnement de Cloud computing et celles qui attendent l'exécution. Lorsque toutes les tâches sont entrées dans le Cloud computing avec la topologie hiérarchique, le gestionnaire de requêtes (nœud racine) rassemble les tâches nécessitant l'exécution et les stocke dans une file d'attente de travail donnant l'ordre d'exécution pour chaque tâche en fonction des caractéristiques de la tâche. En utilisant la programmation BTO le plus approprié. L'ordonnancement est obtenu selon la demande d'emploi et la priorité de service de chaque tâche. Ainsi, le système peut choisir l'ordre d'ordonnancement les plus

appropriés pour réduire le temps d'attente dans la file d'attente et le temps d'exécution pour entrer dans un système.

- **EOLB scheduling**

Le EOLB(Enhanced OLB) proposé combine une OLB(Opportuniste Load Balancing) traditionnelle et le seuil de gestionnaire de services, qui distribue non seulement la tâche de gestionnaire de services le plus approprié en fonction de la propriété de tâche nécessitant l'exécution, mais maintient également l'avantage de OLB traditionnel pour atteindre l'équilibre de charge.

- **EMM scheduling**

Dans la troisième phase, un EMM (Enhanced Min Min) est proposé qui combine une programmation Min-Min et un seuil de nœud de service qui garantira la tâche se voit attribuer à un noeud approprié pour procéder à l'exécution minimum de temps. EMM peut choisir le meilleur nœud de service et utilise ensuite le seuil de noeud de service pour garantir que le nœud exécute la tâche dans les plus brefs délais. Ainsi, le travail peut être réparti de manière efficace et la meilleure allocation des ressources fournie.

L'ordonnancement des trois phases sera employé dans le réseau informatique en nuage hiérarchique. Le calendrier proposé peut planifier la tâche en fonction de la demande des tâches. Ainsi, chaque tâche sera achevée rapidement et efficacement dans le réseau de Cloud computing hiérarchique.

IV.2.3°) Résultats

Les algorithmes de planification distribués tels qu'OLB ou Min- min (MM) sont largement utilisés dans la recherche. Dans cet article, nous comparons les algorithmes suivants: EOLB et EMM planification avec un algorithme de planification OLB et Min-Min (MM). Le tableau5 présente trois scénarios de la conception expérimentale. Chaque script sera combiné avec deux horaires.

Tableau 5 Conception des trois ordonnancements

conception / Planification Script	Sous-directeur de la deuxième phase	Executive-Noeud de troisième phase
Script 1	OLB	MM
Script 2	EOLB	MM
Script 3	EOLB	EMM

La figure 25 montre une comparaison entre makespan: le temps d'exécution total pour toutes les sous-tâches pour chaque script. L'axe y de la figure représente le makespan (cycle de vie d'une tache), tandis que l'axe des x de la figure représente les scripts de la programmation composée de divers ordonnancement, et affiché dans le Tableau 5.Sur la figure 28, le makespan pour le premier script combinant OLB et MM est de 409. Le makespan du second script combinant EOLB et MM est de 260. En outre, le makespan du troisième scénario combinant EOLB et EMM dans cette étude proposée est de 205.

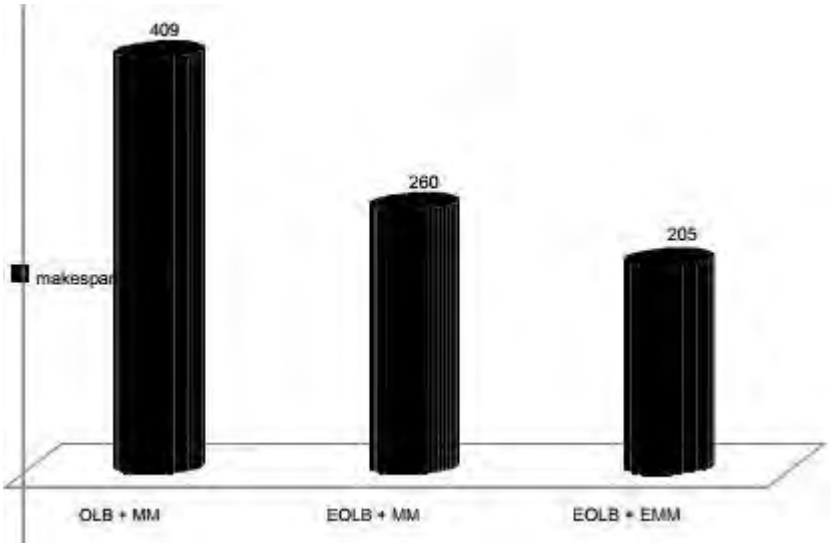

Figure 30 La comparaison des make-span dans chaque ordonnancement

Dans la comparaison des makespan pour les trois scripts, le script proposé est plus rapide que la première avec une différence de temps de 55 et plus rapide que la seconde de 204. La principale raison qui a conduit à cette situation est que le troisième scénario considère non seulement l'équilibre de charge pour chaque nœud, mais utilise également le seuil nécessaire pour éliminer les nœuds inappropriés. Comment juger de la valeur de seuil est un grand défi pour cette recherche. Si la valeur de seuil est trop faible, les ressources peuvent être gaspillées. Si la valeur de seuil est trop élevée, la valeur existera en nom. Comment la valeur de seuil sera décidé dans cette expérience est décrite comme suit: chaque tâche sera autorisé à attendre que le nœud qui peut terminer la tâche dans le temps le plus court alors que le nœud qui peut avoir coûté trop de temps pour exécuter la tâche sera filtrée. En utilisant l'algorithme proposé, le système atteindra les performances d'exécution le plus élevé et atteindre l'équilibre de charge des nœuds.
Les résultats de la simulation pour les trois scripts dans les deux expériences conçues ont prouvé que la méthode d'ordonnancement des tâches combinant EOLB et EMM est plus efficace que d'autres approches de planification réduire le temps d'exécution d'une tâche. Elle est supérieure à l'équilibre de la charge de nœuds, le makespan de toutes les sous-tâches, et à améliorer les performances d'exécution du système.

IV.3°) Présentation de la troisième solution

Le but de cette étude est de montrer quel est l'algorithme d'ordonnancement (FCFS, Round-Robin, Short First) qui respecte le délai (Dead-line) d'une manière efficace. De déterminer l'algorithme d'ordonnancement entre Min-Min et Max-Min celui Qui réalise le « MakeSpan » minimal. Enfin de donner l'algorithme le plus optimal en procédant par une Comparaison en terme de Makespan entre les algorithmes déjà cités et celui proposé appelé optimal.

- ❖ **Résultats**
- ➢ **Critères « Dead-line »**

Si on prend en considération le nombre des tâches terminées avec succès avant le délai (critère : « **Deadline** »), l'algorithme d'ordonnancement **Short First** est plus efficace pour terminer les tâches avant l'arrivée de l'échéance que l'algorithme d'ordonnancement **FCFS**, et **FCFS** est meilleur que l'algorithme d'ordonnancement **Round-Robin**. Nous comparons le nombre de tâches accomplies par chaque algorithme.

Dans la figure 2. Nous pouvons voir les résultats de l'expérience. Sur l'axe des abscisses, il indique le nombre des taches et l'axe des y indique le nombre de tâches terminées avec succès avant que l'échéance ne soit atteinte. En examinant la **figure 31**, nous pouvons dire que l'algorithme **Short First** est les plus efficaces parmi ces algorithmes pour respecter le « **Deadline** ». En augmentant le nombre de tâches nous pouvons remarquer une plus grande différence et conclure que **Short First** est meilleur que **FCFS** et **FCFS** est meilleur que **RounRobin.**

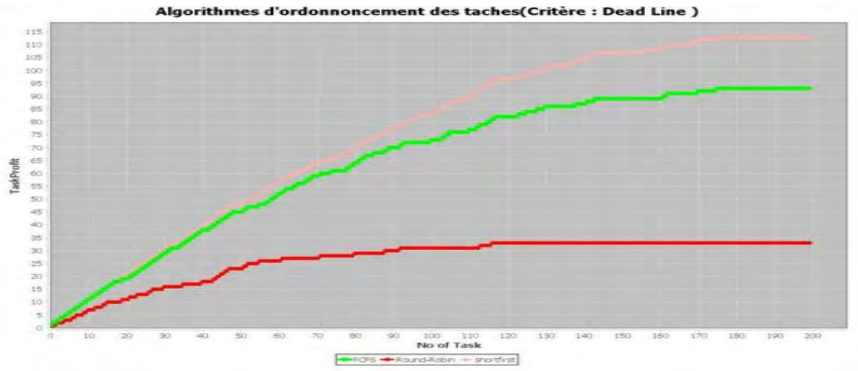

Figure 31 Comparaison sur la base du profit de la tâche avec les 3 algorithmes (Short First, FCFS, Round-Robin)

❖ Critère « makespan » (Min-Min et Max-Min)

Si on prend en considération le « MakeSpan » le temps passé par le système pour réaliser son travail, l'algorithme d'ordonnancement Max-Min est plus efficace pour terminer le travail (toutes les tâches) dans une durée minimal que l'algorithme d'ordonnancement Min-Min. Dans la figure 4, nous pouvons voir les résultats de l'expérience. Sur l'axe des abscisses, il indique le FinshTime et l'axe des y indique le « No of Task » En examinant la figure, nous pouvons dire que l'algorithme Max-Min est le plus efficace que Min-Min pour réaliser un « MakeSpan minimal ». On peut conclure que Max-Min est meilleur que Min-Min si on prend en compte le critère « MakeSpan ».

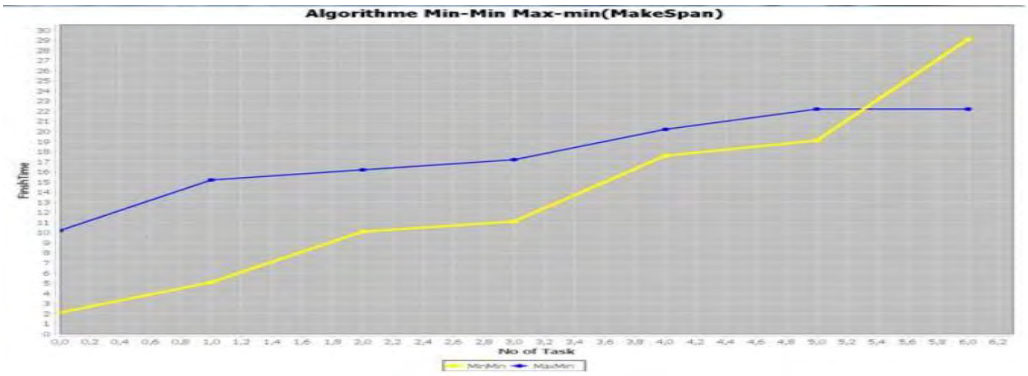

Figure 32 Comparaison selon le critère MakeSpan des algorithmes Max-Min et Min-Min

❖ Critère « makespan » (Min-Min, Max-Min et Optimal proposé)

Après l'exécution de la simulation avec ces trois algorithmes **Max-Min** et **Min-Min** et **l'algorithme proposé** les résultats suivants sont affichés.

L'algorithme Min-Min

Tableau 6 L'attribution des cloudlets sur les ressources par Min-min

Vm id	Cloudlet Id	Time	StarTime	FinshTime
0	1	10	0,1	10.1
1	5	4.33	0.1	4.43
	0	8.33	4.43	12.76
2	4	3	0.1	3.1
	3	3.75	3.1	6.85
	6	7	6.85	13,85
	2	7.5	13.85	**21.35**

On remarque que le Make-span = 21.35

L'algorithme Max-Min

Tableau 7 L'attribution des cloudlets sur les ressources par Max-min

Vm id	Cloudlet Id	Time	StarTime	FinshTime
0	0	12.5	0.2	12.7
1	6	9.33	0.2	9.53
	3	5	9.53	14.53
	4	4	14.53	**18.53**
2	2	7.5	0.2	7.7
	1	5	7.7	12.7
	5	3.25	12.7	15.95

On remarque que le **Make-span = 18.53**

L'algorithme Optimal

Vm id	Cloudlet Id	Time	StarTime	FinshTime
0	2	15	0.3	15.3
1	1	6.67	0.3	6.97
	3	5	6.97	11.97
	5	4.33	11.97	16.3
2	6	7	0.3	7.3
	0	6.25	7.3	13.55
	4	3	13.55	**16.55**

Tableau 8 L'attribution des cloudlets sur les ressources par l'algorithme proposé(Optimal)

On remarque que le **Makespan = 16.55**

On remarque que les résultats de l'algorithme qu'il a proposé donne un makespan plus petit (optimal) par rapport aux autres algorithmes Max-Min et Min-Min.

Ci-dessous est représenté le graphe de Comparaison selon le critère MakeSpan des trois algorithmes

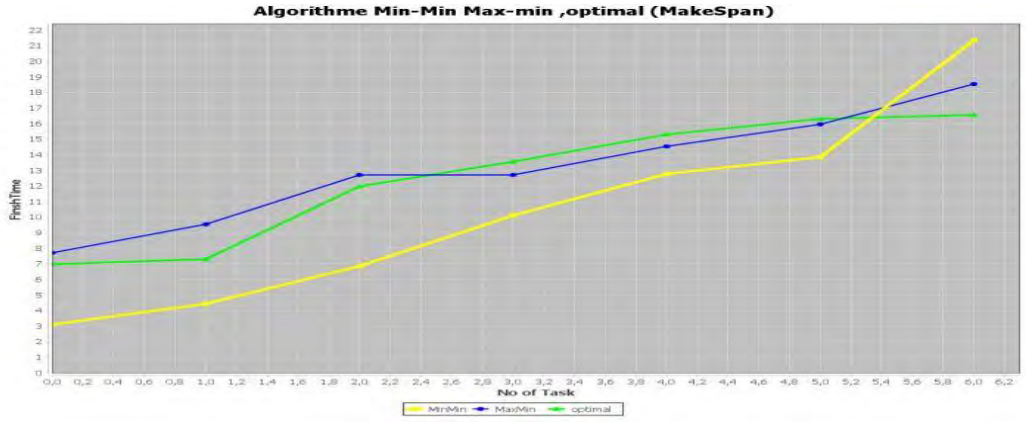

Figure 33 Comparaison selon le critère MakeSpan les trois algorithmes Max-Min, Min-Min et l'algorithme qu'il propose (optimal)

❖ Études comparative des trois solutions

Tableau 9 Étude comparative de solutions d'allocations de ressources

	Article1(Lei Zhao, Jiadai Wang, Jiajia Liu, and Nei Kato)	**Article2**(Shu-Ching, Wang , Kuo-Qin, Yan, Shun-Sheng, Wang, Ching-Wei, Chen)	Article3(Kebir Fatima Zohra)
Type d'algorithme	**EOERA,CHERA et GERA** (Based-Load Balancing and SLA)	**OLB,EOLB,Min-Min,EMM** (Based-Load Balancing)	**FCFS, RR, SJF, Max-Min, Min-Min** (Based-Load Balancing and Quality of Service)
Nature	Dynamique	Dynamique	Statique
Temps de Réponse	Peu de temps	Moyen	Plus de temps pour certains taches
Niveau d'hiérarchisation	Sur tous les niveaux	Sur tous les niveaux	Sur le nuage central
Architecture	3-Tiers	3-Tiers	A deux(2) niveaux