Rapport de Projet : Allocation des ressources dynamiques dans le cloud

NET 4203

Étudiants:
ALLEMAND Fabien
SIX Valentin

1) Description et modélisation du système étudié

Description du système

Le problème que nous étudions est un problème d'allocation de machines virtuelles (VMs) dans un serveur physique d'un cloud. Pour l'étude du système, nous prenons en compte la charge et le nombre de VMs activées. Pour agir sur le système, nous avons la possibilité d'activer ou de désactiver certaines VMs

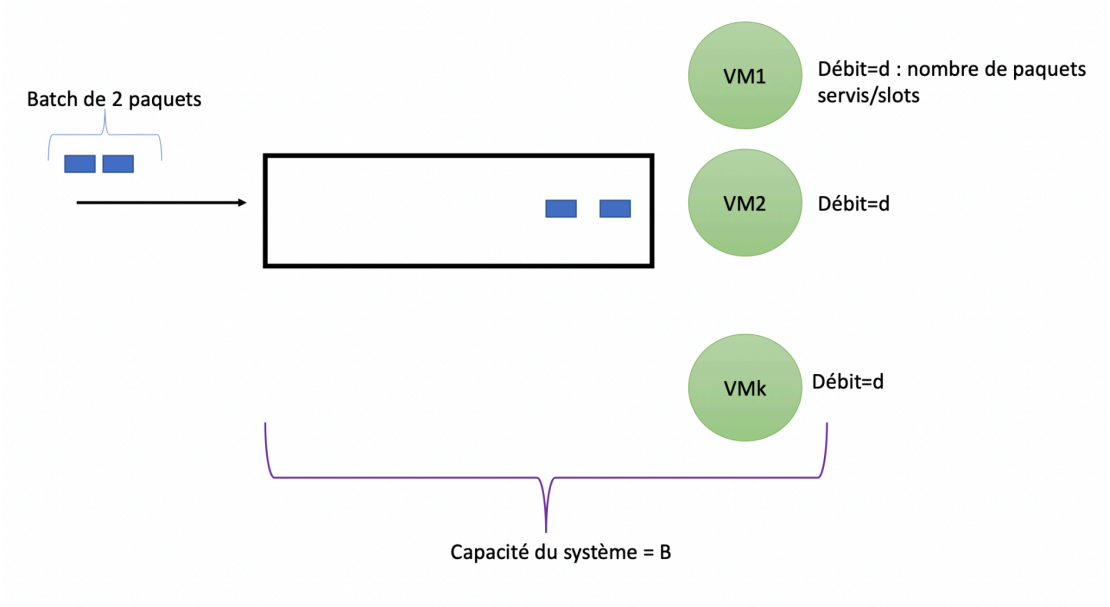


Figure 1: Représentation de notre système

Comme indiqué sur la Figure 1, le système physique présente au maximum K VMs actives. Nous considérons que chaque VM possède le même débit d qui correspond au nombre de paquets servis par slot. Le système possède une capacité de B correspondant aux nombre de clients dans la file d'attente et dans le système (nombre de clients dans le système = nombre de VMS).

En ce qui concerne les arrivées de paquets, on considère qu'ils arrivent par groupe. C'est-à-dire que pour chaque slot, on observe une arrivée d'un groupe de paquets. La probabilité d'une arrivée d'un groupe de i paquets est $P_A(i)$ et b est le nombre maximum de paquets par arrivées.

Concernant les actions que nous pouvons effectuer, nous avons mentionné que nous pouvons activer ou désactiver certaines machines virtuelles. Pour modéliser ces actions, nous noterons α_j (j étant compris entre 0 et K), l'action de maintenir j VMs en activité.

Modélisation du système

Maintenant que nous avons une description précise du système, nous pouvons le modéliser pour effectuer des simulations. Ce système peut être modélisé par une MDP (Markov Decision Process). Pour modéliser les états du système: on notera $s(m,n)$ l'état du système où m est le nombre de clients dans le système et n est le nombre de VMs actives.

Le passage d'un état $s(m,n)$ à un état $s'(m',n')$ par l'action α_j avec la probabilité $P_A(i)$ est caractérisé par:

$$m' = min(B,\ max(0,\ m + i - \sum_{l=1}^{j} d_l))$$

$$n' = j$$

Tout comme un serveur physique réel qui est coûteux à maintenir en activité (coûts énergétique et financier), le système est soumis à des coûts. Voici la description des différents coûts relatifs aux actions et aux états du système:

C_f : coût d'une VM en fonctionnement par slot

C_a : coût d'activation d'une VM

C_d : coût de désactivation d'une VM

C_h : coût de maintien d'un client dans la file

On peut ainsi définir le "gain" obtenu en partant de l'état s et en appliquant l'action α_j :

$$r(s,\alpha_j) = j * C_f + (j - n) * 1_{j>n} * C_a + (n - j) * 1_{n>j} * C_d + (m - j) * C_h$$

Remarque: Nous avons modifié la fonction de coût au niveau du maintien de clients dans la file de façon à ne prendre en compte que les clients qui qui sont réellement en attente. Les clients utilisant déjà les VMs sont pris en compte dans le coût $(j * C_f)$.

Dans le cas de cette étude le gain (tel qu'il est défini dans le cadre de l'apprentissage par renforcement) correspond au coût total généré par le système. Cependant, contrairement au gain classique, on cherchera ici à minimiser le coût total du serveur.

Le système étudié peut être contrôlé par une politique qui définit pour chaque état possible la meilleure action à réaliser. Le but de cette étude est d'obtenir une politique optimale: une politique qui permet de minimiser les coûts du système.

Dans notre étude on fixe arbitrairement les paramètres intrinsèques au système :

- Nombre maximal de VMs: $K = 3$
- Capacité du système: $B = 3$
- Débit: $d = 5$

- Distribution de probabilités d'arrivée de paquets: $P_A = [0.2,\ 0.4,\ 0.3,\ 0.1]$ respectivement pour l'arrivée de 0,1,2 et 3 paquets
- Tous les coûts valent 1

Avec ces paramètres, il existe 16 états distincts et l'agent peut choisir parmi 4 actions.

2) Algorithmes implémentés

Afin d'obtenir la politique optimale pour répondre aux besoins du modèle, nous avons implémenté trois algorithmes d'apprentissages par renforcement: Q-Value, Q-Learning et SARSA.

Notions générales

Remarque: Généralement, les problèmes traités par apprentissage par renforcement visent à maximiser un gain. Dans le cas de cette étude, nous cherchons à minimiser le coût. Par la suite, nous adaptons le vocabulaire et les algorithmes au problème étudié.

Les trois algorithmes étudiés reposent sur la notion de Q valeur: un tableau contenant pour chaque couple état-action la valeur du coût que l'on peut espérer recevoir. Ces algorithmes permettent de trouver de façon itérative la Q valeur optimale pour un problème donné en utilisant la matrice de transition (matrice contenant pour chaque couple état-action la probabilité d'arriver dans chaque autre état) et une fonction de coût (fonction qui associe un coût à chaque action réalisée depuis un état donné).

Algorithme Q-Value

Le premier algorithme implémenté repose sur l'équation de Bellman issue du domaine de la programmation dynamique. Cette équation exprime le coût total d'une action en fonction du coût immédiat de l'action et des potentiels coûts des futures actions.
Dans le cas du modèle étudié, chaque entrée de la matrice Q est mise à jour en considérant que le coût d'une action est constitué de son coût immédiat défini par la fonction de coût et la somme des coûts minimaux qu'il est possible d'obtenir depuis l'état s' pondérés par la probabilité de passer de l'état s à l'état s' en effectuant cette action.
Un facteur de réduction est utilisé pour ajuster l'importance des futurs coûts dans le coût total d'une action choisie (pondération de la somme des coûts futurs potentiels).
La matrice Q est ainsi optimisée de façon itérative jusqu'à ce que l'algorithme converge vers la matrice Q optimale.

$$Q_{n+1}(s,a) = r(s,a) + \gamma \sum_{s' \in S} P^{a}(s,s')\, min(Q_n(s',a'))$$

Algorithme Q-Learning

Plutôt que de calculer la fonction Q de façon calculatoire, l'algorithme Q-Learning cherche à placer le système en conditions réelles et à apprendre la fonction Q en fonction des résultats obtenus. On introduit pour cela la notion d'épisode: scénarios au cours desquels le système agit en fonction Q et optimise Q en fonction du résultat obtenu.

Etant donné que le système étudié ne peut pas arriver dans un état bloquant, on fixe à l'avance la durée des scénarios.
L'algorithme Q-Learning est dit *off-policy* car la politique utilisée pour l'apprentissage (politique utilisée pour simuler le système dans un scénario, dans l'implémentation choisie: politique ε-greedy) n'est pas la même que la politique utilisée pour optimiser la matrice Q (meilleure action possible en étant dans le nouvel état: recherche du minimum).

$$Q(s,a) = Q(s,a) + \alpha * [r(s,a) + \gamma\, min(Q(s',a')) - Q(s,a)]$$

On remarque l'utilisation de deux paramètres dans l'équation de mise à jour de Q:

- γ, le facteur de réduction qui permet (comme dans l'algorithme Q-Value) de pondérer l'importance des coûts futurs.
- α, le facteur d'apprentissage qui sert à pondérer l'importance de l'action choisie: par exemple, une faible valeur implique une faible variation de la valeur de Q pour un état et une action donnés, c'est-à-dire que l'algorithme apprend peu.

Algorithme SARSA

SARSA est l'acronyme de *State-Action-Reward-State-Action* qui peut se traduire en français par Etat-Action-Récompense-Etat-Action. Cet algorithme, ressemble beaucoup à l'algorithme Q-Learning à la seule différence que SARSA effectue un apprentissage *on-policy*, autrement dit, la politique utilisée pour simuler le système est la même que celle utilisée pour optimiser la matrice Q. Cela se voit dans l'équation de mise à jour de la matrice Q où on ne sélectionne plus la meilleure action possible depuis le nouvel état (celle qui offre le coût minimal) mais l'action indiquée par la politique:

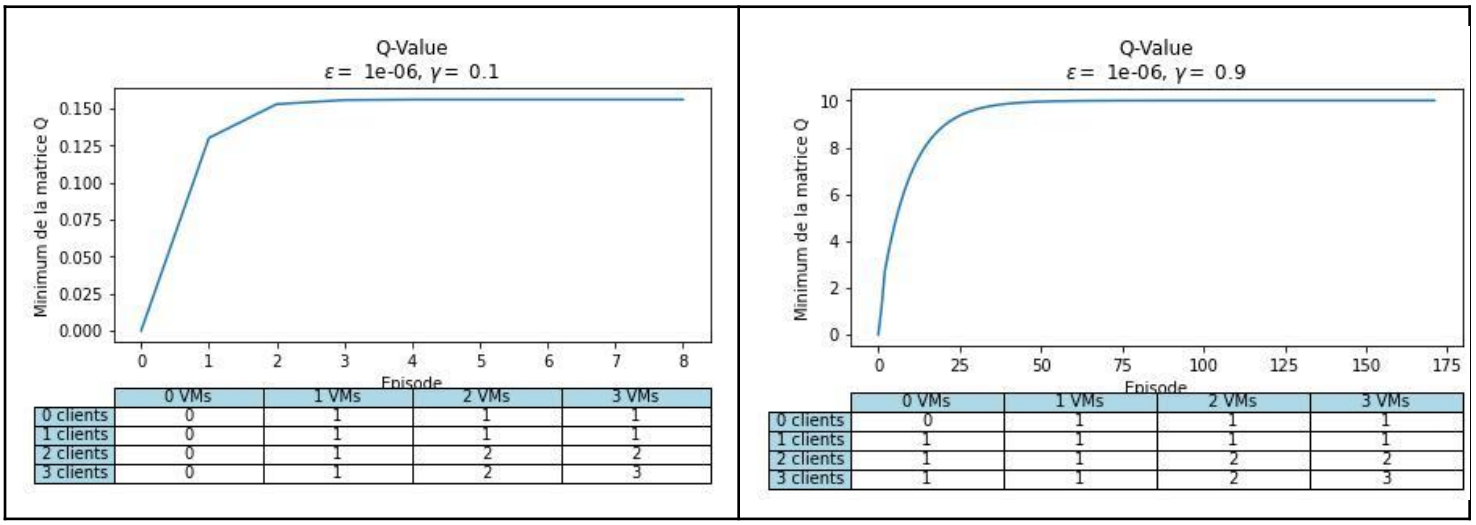
$$Q(s,a) = Q(s,a) + \alpha[r(s,a) + \gamma Q(s',a') - Q(s,a)]$$

On retrouve les paramètres γ et α qui ont la même interprétation (et utilisation) que dans l'algorithme Q-Learning.

3) Comparaison des résultats

Nous cherchons maintenant à appliquer les algorithmes pour résoudre le problème étudié. Pour cela, nous comparons les résultats des différents algorithmes avec différentes valeurs de paramètres.

Pour l'algorithme Q-Value, le paramètre définissant la convergence est fixé à une valeur très faible assurant la convergence de l'algorithme. Il reste un paramètre sur lequel nous pouvons agir: le facteur de réduction. Comme dit précédemment, ce facteur pondère l'impact du coût des potentielles actions possibles suivant l'action choisie. Dans la figure 2, on remarque par exemple qu'avec une valeur de γ trop faible, le système n'allume jamais de VMs quel que soit le nombre de clients en attente et reste dans cet état. Si γ est plus importante, le système allume une VM lorsque des clients arrivent et qu'aucune VM n'est activée ce qui permet par la suite d'allumer d'autres VMs au besoin.



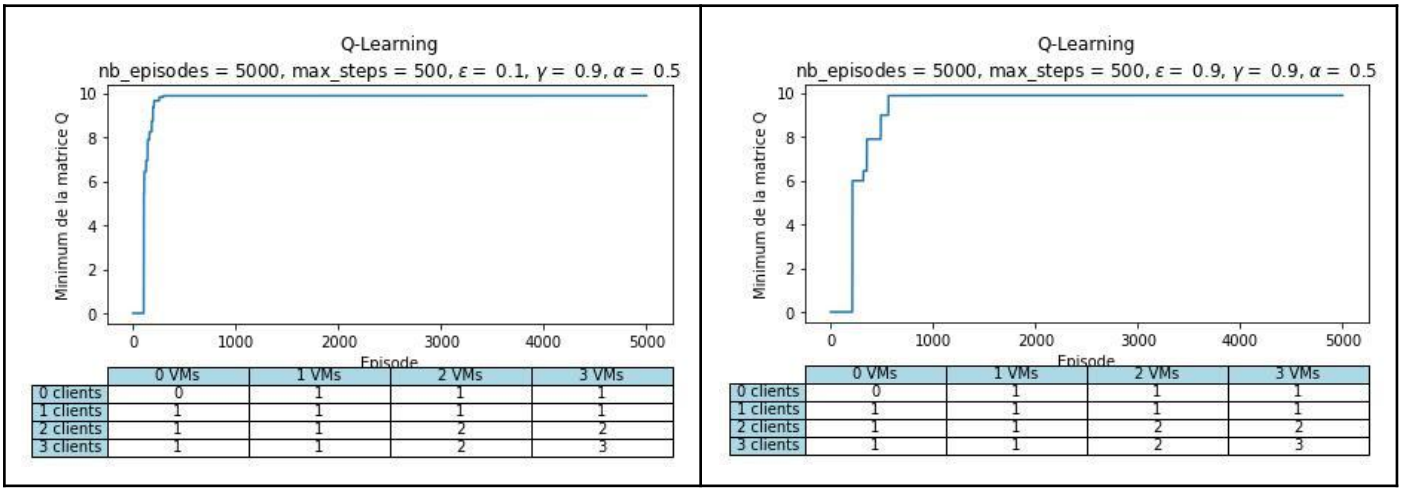
	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	1	1
1 clients	0	1	1	1
2 clients	0	1	2	2
3 clients	0	1	2	3

	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	1	1
1 clients	1	1	1	1
2 clients	1	1	2	2
3 clients	1	1	2	3

Figure 2: Comparaison des résultats de l'algorithme Q-Value pour différentes valeurs de facteur de réduction

Pour le problème étudié une valeur de facteur de réduction égale à 0.9 offre de meilleurs résultats. On conservera donc cette valeur dans la suite de l'étude.

La figure 3 montre l'influence du taux d'exploration pour l'algorithme Q-Learning. On remarque qu'un fort taux d'exploration permet au système d'apprendre de nouveaux comportements qui vont grandement le faire progresser (marches dans la courbe d'évolution des valeurs minimales de Q). Selon nos tests, avec suffisamment d'épisodes l'algorithme converge dans les deux cas mais une valeur plus faible de taux d'exploration donne des résultats plus réguliers (apprentissage plus continu) et une convergence un peu plus rapide.



	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	1	1
1 clients	1	1	1	1
2 clients	1	1	2	2
3 clients	1	1	2	3

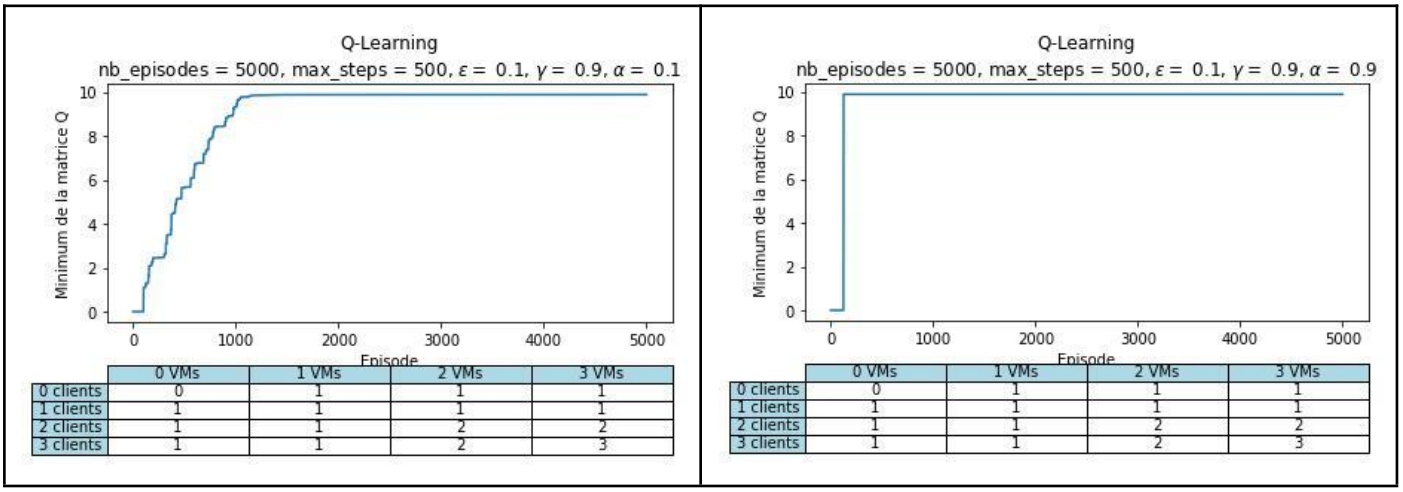
	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	1	1
1 clients	1	1	1	1
2 clients	1	1	2	2
3 clients	1	1	2	3

Figure 3: Comparaison des résultats de l'algorithme Q-Learning pour différentes valeurs de taux d'exploration

Dans la suite, nous utiliserons donc une valeur de taux d'exploration faible ($\varepsilon = 0.1$).

Remarque: Certaines implémentations de l'algorithme Q-Learning font varier la valeur de ε au cours de l'apprentissage afin d'assurer une bonne exploration au début de l'apprentissage et faire plus d'exploitation par la suite.

Le taux d'apprentissage a un fort impact dans notre cas car le système étudié est simple. La figure 4 laisse supposer qu'une valeur plus importante permet d'atteindre d'aussi bons résultats beaucoup plus rapidement. Cependant, dans pour un système plus grand (plus d'états et plus d'actions possibles), un taux d'apprentissage trop fort ne permettrait pas d'apprendre efficacement.

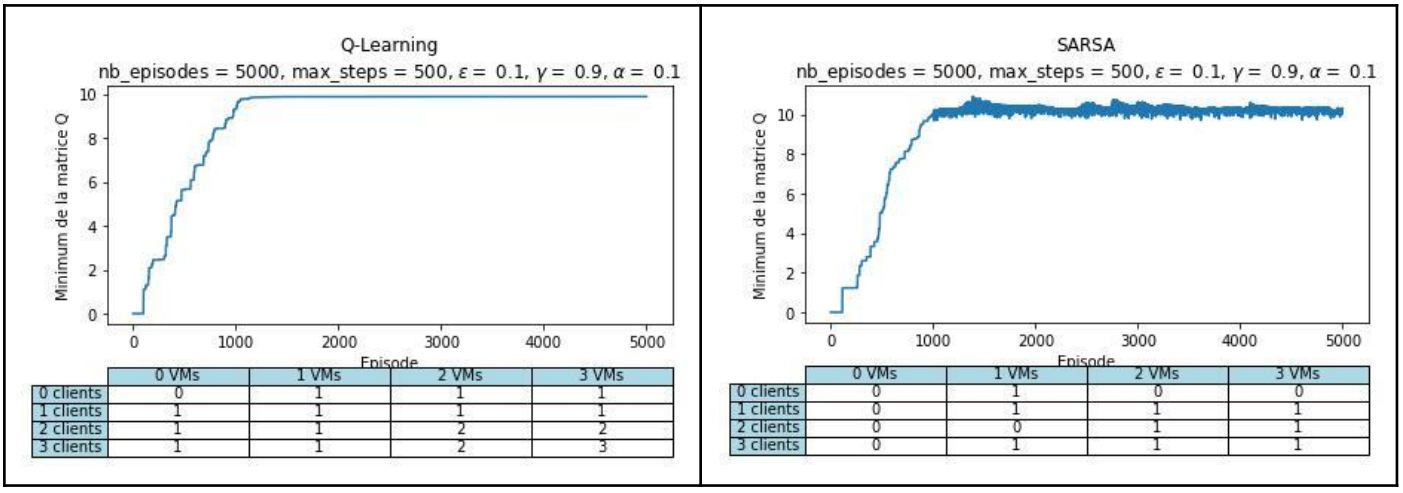


	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	1	1
1 clients	1	1	1	1
2 clients	1	1	2	2
3 clients	1	1	2	3

	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	1	1
1 clients	1	1	1	1
2 clients	1	1	2	2
3 clients	1	1	2	3

Figure 4: Comparaison des résultats de l'algorithme Q-Learning pour différentes valeurs de taux d'apprentissage

Pour les mêmes valeurs de paramètre, les algorithmes Q-Learning et SARSA convergent vers la même valeur en un nombre similaire d'épisodes. Cependant, la politique proposée par SARSA n'est pas satisfaisante pour le système étudié. Par exemple, s'il y a deux clients et deux VMs actives, un des clients sera rejeté du système car la politique conseille de garder une seule VM. L'algorithme Q-Learning semble donc fournir les meilleurs résultats.



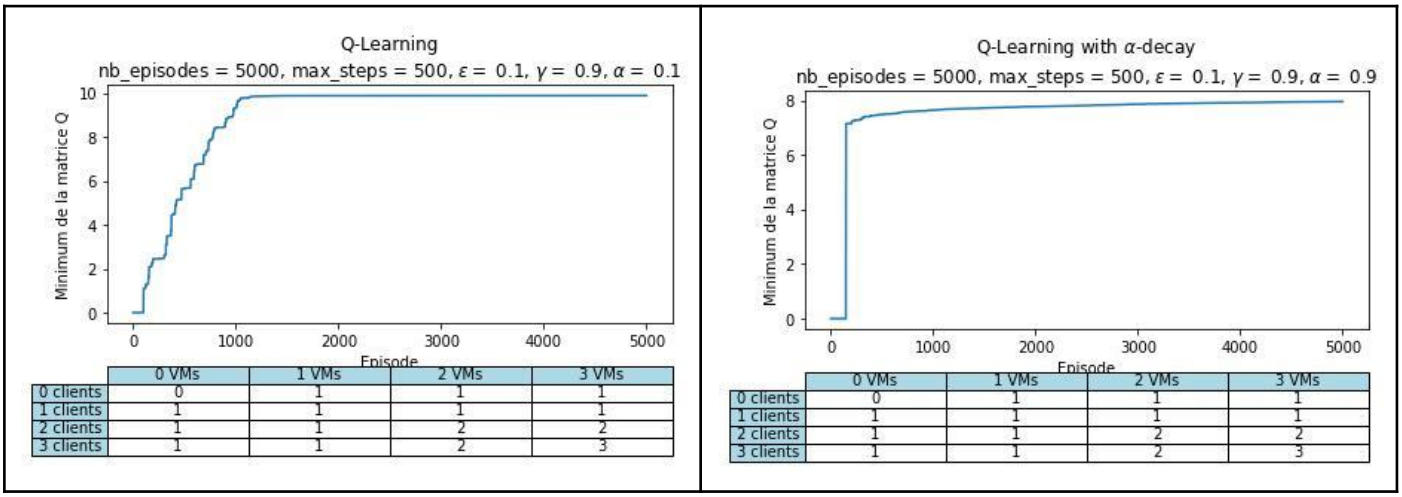
	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	1	1
1 clients	1	1	1	1
2 clients	1	1	2	2
3 clients	1	1	2	3

	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	0	0
1 clients	0	1	1	1
2 clients	0	0	1	1
3 clients	0	1	1	1

Figure 5: Comparaison des résultats de l'algorithme Q-Learning et SARSA

Finalement, nous avons comparé les résultats obtenus avec l'algorithme Q-Learning avec et sans décroissance du taux d'apprentissage au cours du temps. On remarque dans le

second cas que l'algorithme apprend beaucoup sur les premiers épisodes (marche sur la courbe) puis lorsque le taux d'apprentissage diminue, l'apprentissage est plus lent (peu de variation de la courbe).



	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	1	1
1 clients	1	1	1	1
2 clients	1	1	2	2
3 clients	1	1	2	3

	0 VMs	1 VMs	2 VMs	3 VMs
0 clients	0	1	1	1
1 clients	1	1	1	1
2 clients	1	1	2	2
3 clients	1	1	2	3

Figure 6: Comparaison des résultats de l'algorithme Q-Learning avec et sans modification du taux d'apprentissage au cours du temps

4) Conclusion

D'après les différents tests que nous avons effectués, les algorithmes Q-Value et Q-Learning (avec ou sans variation du taux d'apprentissage) renvoient la même politique. Cette politique semble cohérente avec le système étudié contrairement à la politique fournie par SARSA. On peut donc considérer que l'algorithme Q-Learning est le meilleur choix d'algorithmes pour résoudre le problème avec le système choisi car il fournit une politique cohérente et une convergence bien visible tout en fournissant plus de paramètres que Q-Value pour être adapté à d'autres systèmes (systèmes plus complexes qui convergeront plus lentement avec Q-Value).

Cependant, nous avons effectué très peu de tests (limite de temps) et avec un système très simple (puissance de calcul restreinte). Il faudrait effectuer plus de tests, en essayant plus de combinaisons de paramètres (et avec des valeurs moins extrêmes) sur chacun des algorithmes et sur un système plus réaliste pour choisir définitivement un algorithme pour résoudre le problème.