Limiting Memory Reclaiming Impact on VMs Performance

Alexis MERIENNE[1], Dino LOPEZ, and Ramon APARICHIO

[1] Polytech Nice Sophia https://polytech.univ-cotedazur.fr/
[2] Université Côte d'azure http://www.univ-cotedazur.fr/

Abstract. Aujourd'hui, la majorité des datacentres fonctionnent en utilisant des machines virtuelles (VM) qui proposent des services adaptés aux besoins clients. La virtualisation offre aux gestionnaires des datacentres, un moyen d'optimiser les ressources physiques d'un serveur. Dans ce projet, nous allons nous intéresser au partage de la mémoire entre plusieurs VMs d'un serveur. Généralement, une région mémoire est attribuée à une unique VM à la création de celle-ci. Ainsi, la quantité de mémoire utilisée par une VM est statique tout au long de sa phase de vie, ce qui rend l'élaboration de technique d'optimisation complexe. Cependant, il existe déjà des techniques pour récupérer la mémoire utilisée par les VMs. L'une d'elles, qui a été étudiée lors de travaux précédents, est de faire appel à des applications noyaux pour forcer les processus sur lesquels les VMs s'exécutent à lâcher de l'espace mémoire. Sous Linux, ces applications noyaux sont appelées *cgroup*. En revanche, ces opérations ont un effet négatif sur la performance de la VM. Le but de ce projet est de proposer une solution avec laquelle on puisse libérer de l'espace mémoire lorsqu'une VM n'est pas à 100% de ses capacités sans trop affecter ses performances. Ainsi, la décision de savoir combien et quand la mémoire doit être récupérée est un problème qui peut être modélisé par un algorithme de machine learning. Ce modèle est capable de prédire la dégradation du service présent dans une machine virtuelle lorsqu'on applique une diminution de la mémoire allouée. Cette prédiction est utilisée dans le cadre d'un mécanisme qui décide de réduire ou non cette quantité de mémoire allouée.

Keywords: Cloud · Optimisation de mémoire · Performance d'une VM.

1 Introduction

1.1 Contexte

Au cours de la dernière décennie, les technologies de *cloud computing* ont connu un essor considérable, se traduisant par une demande de plus en plus insistante des *datacentres*. Ainsi, l'optimisation des ressources est une question cruciale pour les gestionnaires de ces centres de données. En effet, dans un contexte avec lequel la demande en stockage et gestion des données est en constante croissance et les constructeurs sont soumis à des limites matérielles, il est nécessaire de développer des techniques pour mieux utiliser les ressources en les exploitant à leur capacité maximum. Des travaux sur la gestion des ressources en énergie pour alimenter les centres de données ont déjà été menés, en développant des modèles intelligents pour gérer la consommation électrique des centres de données [3]. Google a par exemple développé son propre agent RL de gestion énergétique de ses *datacentres*. [4]

1.2 Optimisation de la mémoire

Notre projet se concentre sur une gestion système des ressources disponibles dans les centres de données. Plus particulièrement, nous voulons développer une stratégie d'allocation mémoires intelligentes pour les VMs présentes sur un serveur. Il existe déjà des techniques d'optimisation de la ressource mémoire. Les deux principales présentes dans la littérature sont *Ballooning* et *Swapping*. Le *Ballooning*, permet à un hôte physique d'affecter temporairement la mémoire inutilisée d'une VM invitée à une autre grâce à un "Ballon driver". [9]. Ce "Ballon driver" est présent dans chaque VM du nœud et verrouille la mémoire non utilisée par celle-ci. Ce driver communique ensuite avec l'hyperviseur pour allouer cette mémoire à une autre VM. Cette méthode peut engendrer des pertes de performance si le "Ballon driver" est mal calibré et libère trop de mémoire, ce qui fait que la VM n'a plus la quantité requise pour s'exécuter correctement. Le *Swapping* consiste à réduire la mémoire allouée à la VM par le biais de l'hyperviseur, ce qui permet de transférer les adresses mémoires attachées au processus de la VM vers un disque de stockage. [6] Cette méthode est généralement utilisé en dernier recours, car elle affecte grandement la performance de la VM, étant donné qu'utiliser la swap est un processus beaucoup plus lent que la RAM.

1.3 L'apprentissage par renforcement

L'apprentissage par renforcement repose sur le principe d'un agent évoluant dans un environnement donné. Cet agent est capable de prendre des décisions en fonction de l'état de l'environnement, et les appliquer sous la forme d'actions en entrainant une modification (transition) dans l'état de l'environnement. Ces actions sont évaluées par une fonction de récompense, qui détermine si cette action est bénéfique. Lorsqu'on développe un modèle d'apprentissage par renforcement, l'objectif est de déterminer la meilleure *policy*, c'est-à-dire, la suite d'actions qui maximise la fonction de récompense.

La différence entre apprentissage par renforcement et apprentissage par imitation est que pour le premier, l'agent apprend en évoluant de manière autonome dans l'environnement. C'est-à-dire que dans un premier temps, ces actions seront aléatoires et au fur et à mesure de l'apprentissage, il apprendra à maximiser sa fonction de récompense. Tandis que pour l'apprentissage par imitation, l'agent apprend un comportement en se fiant à des données obtenues par un agent tiers, dit *expert*, sachant déjà maximiser la fonction de récompense.

2 Stratégies existantes d'optimisation de ressources mémoires par approche intelligente

2.1 Une approche centralisée

Une première approche intelligente d'optimisation de ressources mémoires a été proposée par une équipe de la Wayne State University en 2009 lors de la 6eme conférence internationale sur l'*Autonomic Computing*. [5]. Le contexte de cette approche est d'optimiser les ressources mémoire à un nœud de Machines Virtuelles dans un serveur. C'est-à-dire que l'objectif est de maximiser les performances globales sur l'ensemble des VMs du nœud. Dans ces travaux, l'équipe de chercheurs propose la modélisation MDP suivante :

- **L'ensemble des états** correspond aux configurations possibles des VMs.
- **l'ensemble des actions** est une combinaison de changements des paramètres configurables des VMs.
- **La fonction de récompense** est déterminée à partir du temps de réponse globale des VMs.

À partir de cette modélisation MDP du problème, les auteurs ont élaboré un mécanisme nommé VCONF gérant les configurations des VMs.

Cette approche est centralisée puisque VCONF est un mécanisme par lequel un unique agent apprend une règle pour optimiser globalement les configurations des VMs du nœud. Cette approche présente quelques limitations, qui sont que la *policy* apprise par l'agent dépend du nombre de VMs configurées et que l'ensemble des actions ne présente pas une granularité suffisante pour reconfigurer correctement les VMs.

2.2 Une approche décentralisée

Une seconde approche est proposée dans l'article *Continuous-Action Reinforcement Learning for Memory Allocation in Virtualized Servers* [7], publié en 2019 par une équipe du Barcelona Supercomputing Center en collaboration avec un chercheur de la Norwegian University of Science and Technology. Cette équipe propose un mécanisme décentralisé d'optimisation des ressources mémoires des VMs d'un nœud serveur, nommé *CAVMem*. Ainsi, pour chaque VM, le mécanisme associe un agent qui apprend une règle pour faire des réclamations de ressources au système global.

La formulation MDP de chaque agent RL est la suivante :

- **L'espace d'états** est une combinaison de variables décrivant l'état de la VM ainsi que des variables décrivant l'état général du système. En effet, les états relevant l'état d'allocation mémoire de la VM sont : la fraction de mémoire allouée à la VM par rapport à la RAM totale, ainsi qu'une variable *missrate* décrivant la quantité de mémoire qu'il manque à une VM pour fonctionner de manière optimale. Les variables décrivant l'état général du système sont *average RAM miss rate* qui décrit la moyenne des *missrate* des VM, *target RAM miss rate*, qui décrit la moyenne des demandes de RAM des VM et *totalMemUse*, qui décrit la fraction de la mémoire de l'hôte utilisée pour exécuter les VMs.
- **L'espace d'action** est déterminé par la *réclamation mémoire* (memory bid), qui est la quantité de mémoire que la VM va demander au système selon ses besoins.
- **La fonction de récompense** a été définie pour que le modèle encourage un niveau équitable de performance entre l'ensemble des VMs du nœud. Elle est déterminée par une punition (récompense négative) proportionnelle au manque d'allocation RAM pour une VM.

Selon la définition du MDP ci-dessus, le problème se situe dans un espace d'action continue. Une implémentation d'algorithme RL en espace d'action continue est Deep Deterministic Policy Gradient [10]. C'est cet algorithme qui a été choisi dans le mécanisme d'optimisation d'allocation mémoire CAVMem. Ce mécanisme est comparé à une approche Deep Q-Learning et à une approche statique. Les résultats présentés dans cet article montre que *CAVMem* présente des résultats similaires à une approche Deep Q-learning mais présente un apprentissage plus rapide.

3 Étude préliminaire du comportement d'une VM sous stress mémoire

Un travail préliminaire a été mené pour comprendre le comportement d'une VM lorsqu'on applique un stress mémoire. Ce travail s'est déroulé sous la forme de plusieurs expériences. Ce sont ces expériences qui vont nous permettre de générer les données nécessaires pour l'apprentissage de notre algorithme Deep Learning.

3.1 Cgroup

À la création d'une machine virtuelle dans un datacenter, on définit sa configuration avec une valeur d'allocation mémoire. Cette valeur est statique, c'est-à-dire qu'au cours de la phase de vie de la VM, cette valeur reste inchangée. Notre projet demande de pouvoir allouer ou de réduire dynamiquement de la mémoire à une VM, c'est-à-dire pendant l'exécution de celle-ci. En effet, les applications noyaux Linux *cgroup* permettent de réaliser cette opération. En effet, les *cgroup* sont une fonctionnalité qui permet de gérer les ressources du système. Leur mode de fonctionnement est organisé en groupes qui sont associés à un

processus d'exécution. C'est sur un groupe que l'on peut définir une limite sur une ressource choisie. Les ressources sur lesquelles on peut agir sont le CPU et la mémoire vive. Lorsqu'un groupe est créé, il doit être associé à des sous-systèmes, qui sont les ressources que l'on veut modifier. Ensuite, le groupe est visible à l'emplacement : */sys/fs/cgroup/<sous-système>/*. Les groupes ont une structure hiérarchique en arborescence. Par exemple, pour accéder à un groupe enfant situé dans une hiérarchie directement sous un groupe parent, le chemin est le suivant : */sys/fs/cgroup/<sous-système>/*.

Pour limiter l'allocation mémoire associé à un groupe, cela s'effectue en changeant la valeur *memory.max* du groupe.

3.2 Mesure des informations mémoire d'une VM

L'étude du comportement d'une machine virtuelle sous stress mémoire nécessite de connaître les variations des indicateurs d'utilisation mémoire du système. Nous avons déterminé qu'il était pertinent de s'intéresser à quatre indicateurs.

- *memused*, la quantité de RAM utilisée dans le serveur.
- *swapused*, la quantité de mémoire en swap.
- *memproc*, la quantité de RAM utilisée par la VM du point de vue de l'hyperviseur.
- *memvm*,la quantité de RAM utilisée par la VM du point de vue de la VM.

Les indicateurs *memused*,*swapused* et *memproc* sont directement accessibles dans le système de fichier de l'hyperviseur. En revanche, lors de nos expériences, nous avons dû mettre en place une communication TCP entre l'hyperviseur et la VM pour obtenir l'indicateur *memvm*.

3.3 Mesure de la performance de la VM

Pour mesurer la performance de la VM, nous exécutons un serveur web apache sur celle-ci. Ensuite, depuis l'hyperviseur, nous questionnons ce serveur. Le temps de réponse que l'on obtient nous donne l'information sur la performance de la VM.

La commande que nous utilisons pour questionner le serveur web est une commande de benchmarks issue de l'API utilitaire d'apache [**?**] :

ab -n NOMBRE_DE_REQUÊTES -c NOMBRE_DE_REQUÊTES_EN_PARALLÈLE

3.4 Protocole expérimental

Nous avons mené nos expériences sur une machine virtuelle ayant pour OS la version Ubuntu 20.04, avec 2 CPU alloués à la VM, 15 Gb d'espace sur disque et initialement 2 Gb de mémoire. Sur cette VM, nous exécutons le serveur HTTP apache.

Les mesures se déroulent selon les étapes suivantes :

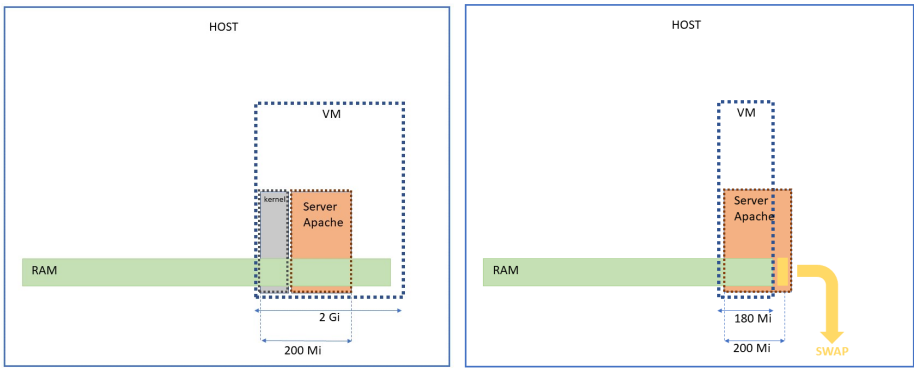
Soit T, le temps total de la mesure.

t=0 : on lance les mesures des quatre indicateurs cités ci-dessus.
t=T/3 : on applique une restriction cgroup telle que *limitcgroup = memproc*
*t=2*T/3* : applique une restriction cgroup telle que *limitcgroup = p * memproc*, avec p le pourcentage de restriction que l'on veut appliquer à la VM.

Pendant tout le temps T de la mesure, nous lançons des benchmarks sur le serveur pour obtenir en continu la performance de la VM.

3.5 Résultats et Observations

Nous avons pu déterminer que le noyau assure la continuité du fonctionnement de la machine virtuelle par un mécanisme de swapping. *(cf. figure 1)* Ce mécanisme ne semble pas prendre en compte l'importance des adresses mémoire qu'il transfère de la RAM vers le disque. En effet, les performances de la machine virtuelle diminuent fortement lorsqu'on diminue la quantité de mémoire disponible pour la VM à une valeur inférieure de ce qu'elle a besoin pour exécuter ses programmes et ses processus noyaux. *(cf. figure 2)* Nous avons également pu trouver une valeur limite en dessous de laquelle la VM cesse de fonctionner.



(a) Système mémoire d'une VM avant restriction

(b) Système mémoire d'une VM après restriction

Fig. 1: Mécanisme swapping VM

4 Élaboration d'un mécanisme d'optimisation intelligent de la mémoire

4.1 Modèle Deep Learning

Les travaux présentés à la section 2 font états de techniques d'optimisation relative entre différentes VMs d'un même nœud sur un serveur. Les règles que

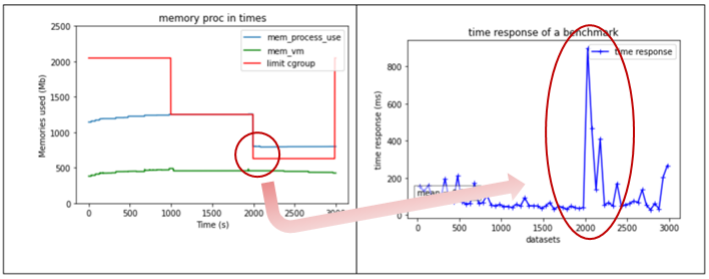


Fig. 2: Observation pic de baisse de performance

les auteurs cherchent à apprendre par technique RL se restreignent à transférer des allocations mémoire d'une VM à une autre.

Notre projet est différent en cela que nous cherchons à élaborer un mécanisme permettant d'optimiser les ressources allouées à une VM en fonction de sa demande au cours du temps.

Formuler le problème sous forme de séries temporelles La performance d'une machine virtuelle dépend de l'action qu'on effectue sur cette machine virtuelle à un instant *t+0*, mais dépend également des actions qui ont été effectués avant ce *t+0*. Ainsi, pour prédire une baisse de performance d'une VM, il faut connaître les états passés de cette VM. Une approche simple RL sans prise en compte des dépendances temporaires ne suffit donc pas pour répondre à la problématique.

Nous avons ainsi développé un modèle Deep Learning RNN (Recurrent Neural Network), capable de prédire un temps de réponse en fonction de l'historique d'activité relatif aux fluctuations de mémoires d'une VM et de l'état actuel de celle-ci.

Plus particulièrement, nous avons développé un modèle capable de prédire une forte baisse de performance lorsque l'on applique une restriction *cgroup*.

Notre idée est d'entraîner un modèle Deep Learning composé de cellules temporelles pour estimer le temps de réponse d'une VM en fonction de son état actuel et passé. Ce temps de réponse permet de connaître la performance d'une VM qui est une variable essentielle pour élaborer un mécanisme d'optimisation de l'allocation mémoire.

Le Dataset Le dataset est construit à partir des données issues des expériences présentées en 3.4. Nous avons pu obtenir 13 expériences dont les données sont exploitables. Certaines des expériences présentaient des données incohérentes liées à des erreurs lors de l'exécution des scripts. La durée de ces expériences est 3000s. La fréquence d'échantillonnage des indicateurs de mémoires est 0.5 s, tandis que la fréquence d'échantillonnage pour les mesures de temps de réponses

est de 45s. Nous avons choisi cette valeur de 45s, car c'est le temps maximal qu'un benchmark met pour finir de s'exécuter.

Finalement, le dataset contient 78 000 données. Une donnée est décrite par 6 variables : *memuse, memswap, memproc, memvm, limitcgroup, responsetime*

Construction d'un dataset de séries temporelles Ce dataset tel quel ne peut pas être utilisé pour entraîner notre modèle. Il est nécessaire de passer par une phase de preprocessing pour construire les séries temporelles que l'on renseignera en entrée de notre modèle.

Ce preprocessing consiste à construire des séries temporelles de taille n selon une méthode dite de *fenêtre glissante*. C'est-à-dire que pour une dataset de taille N, nous obtiendrons un sous-dataset de séries temporelles de taille N-n.

Pour n=*3* et N=*10*, on obtient :

$$[x_0, x_1, x_2, ..., x_9] \longrightarrow [[x_0, x_1, x_2], [x_1, x_2, x_3], ..., [x_7, x_8, x_9]]$$

n est un paramètre de notre modèle que nous avons fait varier lors de la phase du fine-tunning.

De même, nous avons construit deux dataset de séries temporelles. L'un avec, pour chaque donnée, quatre indicateurs de mémoires (*memuse, swapuse, memproc, memvm*) et l'autre avec 2 indicateurs de mémoires (*memproc, memvm*). En effet, nous voulions tester si ajouter des indicateurs d'utilisation de mémoire de l'hyperviseur (*memuse,swapuse*) est pertinent ou non.

On note *NOMBRE_FEATURES* le paramètre associé à cette notion.

Le modèle LSTM Notre modèle présentant les meilleurs résultats contient deux couches LSTM [8] de taille respective (64,32). Une couche de réseau de neurones entièrement connectés de taille 1 à la sortie du modèle permet d'estimer la valeur de temps de réponse. *(cf. figure 3)*

La fonction de loss est *mse*, qui calcule l'erreur entre la valeur prédite et la valeur attendue.

Nous avons entraîné notre modèle avec une répartition de 80 % de données d'entraînements issues du dataset, 20 % des données sont dédiées aux tests.

Pour mesurer la précision du modèle, nous utilisons la métrique *mse* ainsi qu'une observation de l'allure de la courbe obtenue. Nous observons ainsi si le modèle est capable de prédire le pic d'augmentation du temps de réponse lorsque l'on applique une restriction *cgroup*.

Résultats Les paramètres présentant les meilleurs résultats sont n=25 et *NOMBRE_FEATURE*=2. (*cf. figure 4*)

4.2 Mécanisme d'optimisation intelligent de la mémoire (MOIM)

Nous avons construit un mécanisme capable d'optimiser la mémoire requise pour exécuter une machine virtuelle en fonction des prédictions de baisse de performances obtenues grâce à notre modèle LSTM.

Layer (type)	Output Shape	Param #
input (InputLayer)	[(None, 30, 5)]	0
lstm_layer_1 (LSTM)	(None, 30, 64)	17920
lstm_layer_2 (LSTM)	(None, 32)	12416
dropout_2 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 1)	33

Total params: 30,369
Trainable params: 30,369
Non-trainable params: 0

Fig. 3: Résumé du modèle LSTM

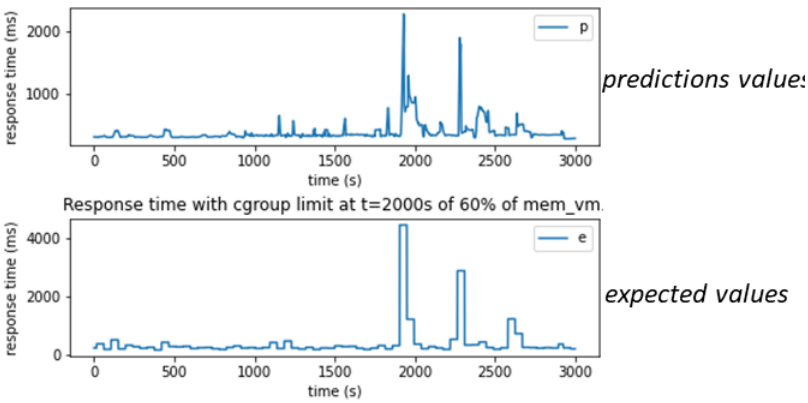


Fig. 4: Résultat de prédiction du modèle

Ce mécanisme mesure les indicateurs de mémoire avec une même fréquence d'échantillonnage que pour constituer le dataset. De même, il effectue des benchmarks sur le serveur apache HTTP de la VM pour obtenir son temps de réponse.

Dès lors qu'il obtient une nouvelle valeur de temps de réponse, c'est-à-dire qu'un benchmark ait fini de s'exécuter, alors il constitue une série temporelle contenant les indicateurs de mémoire de taille n, la limite *cgroup* appliqué à la VM et le temps de réponse obtenu. On note t le temps auquel le mécanisme exécute cette étape.

Cette série temporelle est proposée en entrée de notre modèle LSTM. La prédiction du modèle est le temps de réponse à un temps *t+1*.

C'est cette prédiction qui permet de déterminer l'action à effectuer.

Mécanismes d'actions Pour cette partie, nous nous sommes inspirés des algorithmes de congestions TCP. Ces algorithmes consistent à augmenter progressivement la fréquence d'envoi de paquets via protocole TCP jusqu'à ce qu'on arrive à une fréquence d'envoi où le réseau n'arrive plus à prendre charge le traitement des paquets. (congestion). On diminue alors par deux la fréquence d'envoi.

Dans notre cas, nous diminuons progressivement la mémoire allouée à la VM, avec un pas égal à 90 % de la dernière limite cgroup, jusqu'à ce que la prédiction du temps de réponse dépasse un seuil.

Nous avons en réalité deux seuils. Si la prédiction dépasse le premier, alors on ne fait rien. Si elle dépasse le second, on augmente fortement la mémoire allouée.

En effet, le premier seuil représente un état de la VM où l'allocation mémoire est optimisée, c'est-à-dire que l'on a suffisamment libéré de la mémoire sans trop dégrader la performance du système. Le second seuil représente un état de la VM où l'allocation mémoire est trop faible. C'est-à-dire que l'on a libéré trop de mémoire et que la performance du système en est affectée.

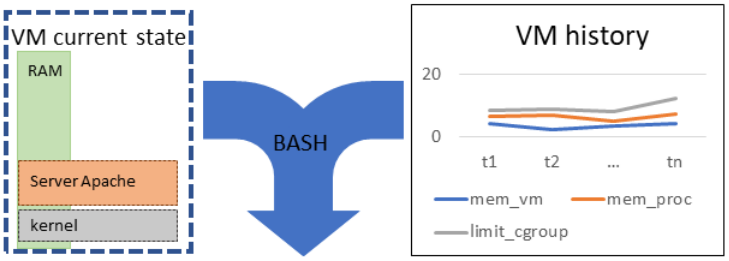
Calcul des seuils Les seuils sont calculés en fonction de la moyenne des temps de réponse effectivement observés. Cette moyenne est calculée sur les dix derniers temps de réponses obtenus.

Le premier seuil est égal à 1.1 de cette moyenne. Le second seuil est égal à 1.5 de cette moyenne.

4.3 Résultats

Notre mécanisme parvient à limiter dynamiquement la mémoire utilisée par la VM sans provoquer de dégradation des performances dans le temps de réponse. On observe également que lorsqu'il se "trompe" et réduit la quantité de mémoire trop radicalement, alors il sait réclamer de la mémoire pour revenir à un état plus stable.

Cependant, dès lors que le système est à un état qui ne correspond pas aux situations observées dans le dataset d'entraînement, alors le comportement du mécanisme devient incertain.




```
response_time_predict = modelLSTM(BASH)
if response_time_predict < 1.1 * AVG(last 10 response time):
    memory.max *= 0.90
if response_time_predict > 1.5 * AVG(last 10 response time):
    memory.max *= 2
else :
    //DO NOTHING
```

Fig. 5: Schéma MOIM

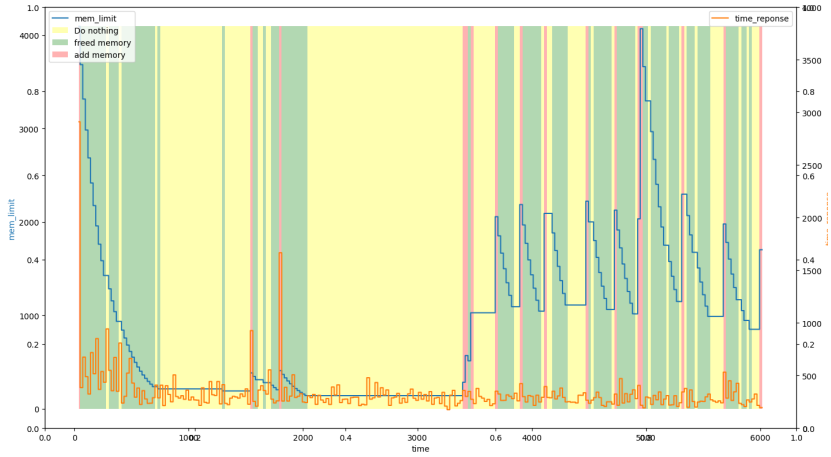


Fig. 6: Mécanisme MOIM résultat

5 Conclusion et discutions

Dans cet article, nous proposons MOIM, un mécanisme utilisant un algorithme de machine learning pour évaluer la performance d'une machine virtuelle lorsqu'on applique une restriction mémoire. Ce mécanisme permet de réduire la quantité de mémoire alloué à une machine virtuelle sans altérer les performances. Ce mécanisme, en l'état, a été entraîné avec des données répondant à un unique situation, qui est une restriction de la mémoire sur une VM présentant un serveur HTTP apache. Des travaux futurs évalueront son fonctionnement avec des données issues d'un plus large panel de situations.

Ainsi, pour étendre ce travail à une application industrielle, il faudrait étudier le comportement de VM sous différentes conditions et exécutant différentes applications.

L'optimisation de ressources physiques dans les datacenters est une thématique enrichissante, car elle répond à des besoins concrets de l'industrie tout en s'inscrivant dans une dynamique d'efficacité énergétique.

Le code utilisé lors de ce travail est disponible ici : https://github.com/AlexisMerienne/TER-2022-033

References

1. Author, Lilian Weng.: Title.: A (Long) Peek into Reinforcement Learning . Publisher, Lil'Log (2018) https://lilianweng.github.io/posts/2018-02-19-rl-overview/
2. Author, Zoltán Lőrincz.: Title.:A brief overview of Imitation Learning. Publisher, Medium (2019). https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c
3. Author, Gille Madi Wamb. Title.:Combiner la programmation par contraintes et l'apprentissage machine pour construire un modèle éco-énergétique pour petits et moyens data centers. Publisher. HAL theses (2017) https://theses.hal.science/tel-01665187/
4. Author, Jim Gao. Title.:Machine Learning Applications for Data Center Optimization. Publisher. Google Publication. https://www.gstatic.com/gumdrop/sustainability/42542.pdf
5. Authors, Jia Rao, Xiangping Bu, Cheng Zhong Xu, Leyi Wang. : Title. In: VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration. Journal,ICAC '09: Proceedings of the 6th international conference on Autonomic computing (2009) https://dl.acm.org/doi/10.1145/1555228.1555263
6. Publisher. The Apache Software Foundation, The Apache Software Foundation
7. Authors, Luis A. Garrido, Rajiv Nishtala, Paul Carpenter,: Title. In: Continuous-Action Reinforcement Learning for Memory Allocation in Virtualized Servers (2019). Publisher, Berlín: Springer https://link.springer.com/chapter/10.1007/978-3-030-34356-9-2
8. Author, Sepp Hochreiter., Author, Jürgen Schmidhuber.: Title. In: Long Short-Term Memory. :Publisher, Neural Computation (1997).: https://direct.mit.edu/neco/article-abstract/9/8/1735/6109/Long-Short-Term-Memory?redirectedFrom=fulltext
9. Author, Jui-Hao Chiang.: Title. Working Set-based Physical Memory Ballooning. Publisher. 10th international Conference on Automic Computing (2013). https://www.usenix.org/conference/icac13/technical-sessions/presentation/chiang
10. Auhtor, Daan Wierstra.: Title. Continuous control with deep reinforcement learning. Publisher. Google Deepmind (2015). https://arxiv.org/abs/1509.02971v6