



République Algérienne Démocratique et Populaire



Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Akli Mohand Oulhadj de Bouira

Faculté des Sciences et des Sciences Appliquées

Département d'Informatique

Mémoire de Master 2

en Informatique

Spécialité

Génie Systèmes Informatiques (GSI)

Ingénierie des Systèmes d'Information et de Logiciel (ISIL)

Thème

Autoscaling Prédictif

Encadré par

— DR.BADIS Lyes

Réalisé par

— M. BOUZIDI Adel

— MLLE. HAMIDOU Imane



République Algérienne Démocratique et Populaire



Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Akli Mohand Oulhadj de Bouira

Faculté des Sciences et des Sciences Appliquées

Département d'Informatique

Mémoire de Master 2

en Informatique

Spécialité

Génie Systèmes Informatiques (GSI)

Ingénierie des Systèmes d'Information et de Logiciel (ISIL)

Thème

Mise à l'Échelle Automatique et Prédictive basée sur
l'Orchestrator Kubernetes (MEAP)

Encadré par

— DR.BADIS Lyes

Réalisé par

— M. BOUZIDI Adel

— MLLE. HAMIDOU Imane

Remerciements

Tout d'abord à ALLAH l'unique dieu.

On tient à remercier nos très chers parents que nul remerciement, aucun mot ne pourrait exprimer à leur juste valeur la gratitude et l'amour qu'on vous porte. On met entre vos mains, le fruit de longues années d'études, de votre amour de votre tendresse, de longs jours d'apprentissage. Votre éducation votre soutien et votre encouragement nous a toujours donné de la force pour persévéérer et pour prospérer dans la vie. Chaque ligne de cette thèse chaque mot et chaque lettre vous exprime la reconnaissance, le respect, l'estime et le merci d'être nos parents que Dieu vous garde.

On tient à exprimer nos sincères remerciements avec un grand plaisir et un grand respect à notre encadreur DOCTEUR BADIS LYES professeur à l'université de Bouira, pour ses précieux conseils, sa disponibilité et ses encouragements qui nous ont permis de réaliser ce travail.

Nous tiendrons aussi à exprimer l'honneur aux membres du jury, en acceptant d'évaluer notre travail. Nous nous acquittons, enfin, volontiers d'un devoir de gratitude et de remerciements à tous nos enseignants pour la qualité de l'enseignement qu'ils ont bien voulu nous prodiguer durant nos études afin de nous fournir une formation efficiente.

On ne peut pas nommer ici toutes les personnes qui de près ou de loin nous a aidé et encouragé mais nous les remercions vivement.

Dédicaces

Ma plus grande gratitude et tout mon amour à mes parents, qui ont su me faire confiance, me soutenir et m'encourager au cours de ma vie.

BOUZIDI Adel

Dédicaces

L'âme de ma chère grand-mère GOUAA Tassaidit, J'aurais tant aimé que vous soyez présents.

Que Dieu ait vos âmes dans sa sainte miséricorde.

Mes chers parents, frères, soeurs et amis.

HAMIDOU Imane

ملخص

الخدمات المصغرة تمثل العديد من المزايا من حيث سهولة النقل وخفة الوزن وسهولة التوسيع مقارنةً بالافتراضية المستندة إلى الأجهزة الافتراضية. إلا أنها تواجه العديد من التحديات ، على وجه التحديد من حيث قابلية التوسيع. في الواقع ، تم تجهيز معظم الانظمة مثل كيوبرنتاس بأنظمة قياس ذاتي تفاعلي. لا يمكن لأنظمة القياس التلقائي التفاعلية هذه أن تضمن جودة الخدمة المطلوبة نظرًا لبطئها في تحجب التحميل الزائد أثناء القياس التفاعلي ، والذي يمكن أن يكون ناتجًا عن القيمة غير الصحيحة لعتبة القياس التلقائي. في هذا العمل ، نقترح طريقة قياس تلقائي تنبؤية جديدة لمجموعة الخدمات المصغرة التي تعمل على نظام كيوبرنتاس. في البداية ، نقترح بنية معيارية. بعد ذلك ، نقوم بدمج مشكلة القياس التلقائي رياضيًا جنباً إلى جنب مع خوارزمية لتسلیط الضوء على التصميم العام. بعد ذلك ، نقوم بتصميم وتنفيذ نموذج تنبؤي يعتمد على طريقة التنبؤ المعروفة باسم شبكة الذاكرة طويلة قصيرة المدى. بعد ذلك ، نقوم بتطوير نظام القياس التلقائي باستخدام نموذج الذاكرة طويلة قصيرة المدى المدرّب ، إضافةً نسخة متماثلة جديدة بشكل استباقي إذا تجاوز الاستخدام المتوقع حدًا معيناً. أخيراً ، نوضح تطور التنبؤ باستخدام منصة تحليل البيانات قرافانا و قاعدة بيانات السلاسل الرمزية انفلاكس ديبي. جنباً إلى جنب مع النتائج التجريبية ، يُظهر النموذج الأولى المطور المسمى بـ التحريم التلقائي والتنبؤي أن التحريم التلقائي والتنبؤي قادر على تحسين الأداء وتجنب التحميل الزائد، مما يضمن وبالتالي جودة الخدمة المطلوبة.

الكلمات الدالة : الحوسنة السحابية ، الخدمات المصغرة ، القياس ، الحاويات ، الذاكرة طويلة قصيرة المدى ، الطرق التنبؤية ، كيوبرنتاس.

Abstract

Microservices represent many advantages in terms of better probability, lightweight, and easy scalability compared to Virtual Machine (**VM**)-based virtualization. They introduce several challenges, specifically in terms of scalability. In fact, most of orchestrators such as Kubernetes are equipped with reactive autoscaling systems. These reactive autoscaling systems cannot guarantee the requested Quality of Service (**QoS**) due to their slowness to avoid overloading while scaling reactively, which can be caused by non correct value of autoscaling threshold. In this work, we propose a novel predictive autoscaling method for microservices running on Kubernetes (K - eight characters - S) (**K8s**) cluster called Mise à l'Échelle Automatique et Prédictive (**MEAP**). Initially, we propose a modular architecture. Then, we mathematically model the autoscaling problem along with a heuristic to highlight the global design. Thereafter, we design and implement a predictive model based on the prediction method Long Short Term Memory (**LSTM**) Encoder-Decoder to predict the future evolution of resources usage. Then, we develop an autoscaling system using the trained **LSTM** model, to proactively add new containers if the predicted use of resources exceeds certain threshold. Finally, we demonstrate the evolution of prediction using Grafana and InfluxDB. The developed prototype **MEAP** along with the experimental results show that **MEAP** is able to improve the performances and avoid overloading microservices, guaranteeing thus the requested **QoS**.

Keywords : Cloud computing, microservices, scaling, containers, **LSTM**, predictive method, Kubernetes.

Résumé

Les microservices représentent de nombreux avantages en termes de meilleure probabilité, de légèreté et d'évolutivité facile par rapport à la virtualisation basée sur les Machine Virtuelle (**MV**). Ils introduisent plusieurs défis, notamment en termes d'évolutivité. En fait, la plupart des orchestrateurs tels que **K8s** sont équipés de systèmes de mise à l'échelle réactifs. Ces systèmes de mise à l'échelle réactifs ne peuvent pas garantir la **QoS** demandée en raison de leur lenteur à éviter la surcharge lors de la mise à l'échelle réactive, qui peut être causée par une valeur incorrecte du seuil de mise à l'échelle. Dans ce travail, nous proposons une nouvelle méthode de mise à l'échelle prédictive pour les microservices exécutés sur un cluster Kubernetes, se référant à nous **MEAP**. Dans un premier temps, nous proposons une architecture modulaire. Ensuite, nous modélisons mathématiquement le problème de mise à l'échelle avec une heuristique pour mettre en évidence la conception globale. Par la suite, nous concevons et implementons un modèle prédictif basé sur la méthode de prédiction **LSTM** Encodeur-Décodeur pour prédire l'évolution future de l'utilisation des ressources. Ensuite, nous développons un système de mise à l'échelle automatique à l'aide du modèle **LSTM** entraîné, pour ajouter de manière proactive de nouveaux conteneurs si l'utilisation prévue des ressources dépasse un certain seuil. Enfin, nous démontrons l'évolution de la prédiction en utilisant Grafana et InfluxDB. Le prototype **MEAP** développé ainsi que les résultats expérimentaux montrent que **MEAP** est capable d'améliorer les performances et d'éviter de surcharger les microservices, garantissant ainsi la **QoS** demandée.

Mots clés : Cloud computing, microservices, mise à l'échelle, containers, **LSTM**, predictive méthode, Kubernetes.

Table des matières

Table des matières	i
Table des figures	vi
Liste des tableaux	viii
Liste des abréviations	ix
1 Introduction Générale	1
1.1 Contexte Général	2
1.2 Problématique et Objectifs	3
1.3 Contributions	4
1.4 Organisation du Travail	6
2 La technologie de conteneurisation Docker	7
2.1 Introduction	8
2.2 Cloud Natif	8
2.3 Les MicroServices	8
2.3.1 Les Avantages des MicroServices	8
2.4 La Virtualisation	8
2.5 La Conteneurisation (Virtualisation par Conteneurs)	9
2.5.1 Ce qui Différencie la Virtualisation de la Conteneurisation	9
2.5.2 Les MicroServices et les Conteneurs	9
2.6 La Technologie des Conteneurs Docker	10
2.6.1 C'est Quoi Docker	10
2.6.2 Architecture et Principaux Composants de Docker	10
2.7 Conclusion	12

3 L'orchestration des conteneurs Kubernetes	13
3.1 Introduction	14
3.2 L'Orchestration des Conteneurs	14
3.3 Le Système d'orchestration Kubernetes	14
3.3.1 Architecture de Kubernetes	15
3.3.2 Composants d'un Cluster Kubernetes	15
3.3.3 Élément Nécessaire Pour le Déploiement d'un Microservice sur Kubernetes	17
3.4 La Mise à l'Échelle Automatique de Kubernetes (Autoscaling)	19
3.4.1 Le Type Mise à l'échelle Horizontale des Pods (HPA)	19
3.4.2 Le Type Mise à l'échelle Automatique des Pods Verticaux (VPA)	20
3.4.3 Le type de Mise à l'échelle Automatique du Cluster	20
3.5 Conclusion	21
4 La méthode de prédiction LSTM	22
4.1 Introduction	23
4.2 La Méthode de Prédiction (LSTM)	23
4.2.1 L'Apprentissage Automatique (Machine Learning)	23
4.2.2 Apprentissage Profond (Deep Learning)	23
4.2.3 Différents Types de Réseaux de Neurones dans Deep Learning	23
4.2.3.1 Les Réseaux de Neurones Artificiels (Artificial Neural Networks (ANN))	24
4.2.3.2 Les Réseaux de Neurones Convolutifs (Convolution Neural Networks (CNN))	24
4.2.3.3 Les Réseaux de Neurones Récursifs (Recurrent Neural Networks (RNN))	25
4.2.3.4 Limites des RNN Simples	26
4.2.3.5 Long Short Term Memory (LSTM)	27
4.2.3.6 Cellule LSTM Pas à Pas	28
4.2.4 Entraîner LSTM	32
4.2.4.1 La Rétro-Propagation du Gradient	33
4.2.5 Modes d'Utilisations de LSTM	35
4.2.6 Types des Modèles LSTM	38
4.2.6.1 Modèles LSTM Uni-Variés avec une Seule Étape (One Step)	38
4.2.6.2 Modèles LSTM Multivariés avec Plusieurs Étapes (Multi-Steps)	39
4.2.7 Préparation des Données pour les Réseaux LSTM	40

4.3 Conclusion	42
5 Travaux connexes	43
5.1 Introduction	44
5.2 Travaux Connexes	44
5.2.1 Proactive Autoscaling for Cloud-Native Applications using Machine Learning	44
5.2.2 Agnostic Approach for Microservices Autoscaling in Cloud Applications . .	45
5.2.3 An Improved Kubernetes Scheduling Algorithm For Deep Learning Platform	45
5.2.4 An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud	46
5.2.5 Adaptive scaling of Kubernetes Pods LIBRA	47
5.2.6 Proactive Autoscaling for Edge Computing Systems with Kubernetes . .	47
5.2.7 A lightweight autoscaling mechanism for fog computing in industrial applications	47
5.2.8 Quantifying Cloud Elasticity with Container-Based Autoscaling	48
5.2.9 A study on performance measures for auto-scaling CPU-intensive containerized applications	48
5.2.10 Building an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows	48
5.2.11 Autoscaling Pods on an On-Premise Kubernetes Infrastructure QoS-Aware	48
5.3 Analyse Comparative	48
5.4 Conclusion	49
6 L'Approche de Mise à l'Échelle Automatique et Prédictive (MEAP) - Concept	50
6.1 Introduction	51
6.2 Architecture Globale de MEAP	51
6.3 Le Modèle Prédictif Choisi	53
6.4 Formulation Mathématique du Problème Correspondant à MEAP	54
6.5 L'Algorithmme de Mise à l'Échelle Automatique et Prédictive MEAP	56
6.6 Conclusion	57
7 Tests et Résultats	58
7.1 Introduction	59
7.2 Plateforme de Mise à l'Échelle Automatique et Prédictive	59

7.2.1	Scénario et Procédure de Mise à l'Échelle Automatique	59
7.2.2	Technologies et Outils Utilisés Pour le Déploiement de MEAP	72
7.2.3	Procédure de Déploiement du Système de Mise à l'Échelle Automatique et Préditive	73
7.3	Résultats	79
7.3.1	La méthode Root Mean Square Error (Root Mean Square Error (RMSE))	79
7.3.2	Évaluation du Modèle LSTM	79
7.3.2.1	Le RMSE du modèle LSTM en Changeant la Taille du Batch d'Entraînement (Batch Size)	79
7.3.2.2	Le RMSE du modèle LSTM en changeant le Nombre d'Époques d'entraînement (epoch) et en Utilisant le Meilleur Batch Size Trouvé	81
7.3.2.3	Le RMSE du Modèle LSTM en Changeant la Taille du Batch d'Entraînement (Batch Size) en Utilisant le Meilleur Nombre d'époques Trouvé	82
7.3.2.4	Le RMSE du Modèle LSTM en Changeant la Taille du Vecteur 'H' en Utilisant les Meilleurs Valeurs d'époques et du Batch Size Trouvées	83
7.3.2.5	Le RMSE du Modèle LSTM en Changeant la Fenêtre de Prédition (TimeSteps)	84
7.3.2.6	Le RMSE du Modèle LSTM en Changeant le taux d'apprentissage α (learning rate)	85
7.3.3	Évaluation de l'approche de Mise à l'Échelle Automatique et Préditive MEAP	86
7.3.3.1	Comparaison de la consommation de Central Processing Unit (CPU) en utilisant l'approche de mise à l'échelle automatique et prédictive et l'approche de mise à l'échelle automatique non prédictive	86
7.3.3.2	Maximum Consommation de CPU en Variant la Valeur du Seuil de Mise à l'Échelle	87
7.4	Conclusion	88
8 Conclusion Générale et Perspectives		89
Bibliographie		91

A Construction des Images : "pas-ss-0" et "build_model"	98
A.1 Introduction	98
A.2 Le Fichier Dockerfile Utilisé pour construire l'Image du Microservice "pas-ss-0" .	98
A.3 Le Fichier Dockerfile Utilisé pour construire l'Image du Microservice "build_model"	99
B Les Fonctions d'Activation Utilisées par LSTM	100
B.1 Introduction	100
B.2 La Fonction d'activation Sigmoïde	100
B.3 La Fonction d'activation tangente hyperbolique (Tanh)	101

Table des figures

2.1	Architecture de Docker	10
2.2	Image Crée en Utilisant Dockerfile	11
3.1	Architecture de Kubernetes	15
3.2	Architecture d'un Pod	17
3.3	Architecture de Réplicaset	18
3.4	Déploiement	19
3.5	(HPA) VS (VPA)	20
4.1	Réseaux Neuronaux Artificiels	24
4.2	Architecture MLP	25
4.3	Tanh	26
4.4	Tanh	27
4.5	LSTM	28
4.6	Architecture d'une Cellule LSTM	29
4.7	Présentation de la porte d'oubli	29
4.8	Présentation de la Porte d'Entrée	30
4.9	Représentation de l'État de la Cellule	31
4.10	Représentation de la Porte de Sortie	32
4.11	Rétro-propagation du gradient	33
4.12	Un à un (One to One)	36
4.13	Un à Plusieurs (One to Many)	36
4.14	Plusieurs à Un (Many-to-One)	37
4.15	Plusieurs à Plusieurs (Many-to-Many)	37
4.16	Archiecture de Mémoire à Long Terme Empilée	38

4.17 Architecture de LSTM Bidirectionnels	39
4.18 LSTM Bidirectionnels	39
4.19 Architecture du Modèle Encodeur-Décodeur	40
4.20 La Fenêtre de Prédiction (TimeSteps)	41
4.21 Nombres d'unités en LSTM	41
4.22 Le Batch Size (Taille du Lot)	42
4.23 Nombre d'Époques (Epoch)	42
6.1 Architecture Globale de MEAP	51
6.2 Architecture du Modèle de Prédiction Utilisé (LSTM Encodeur-Décodeur)	53
7.1 Plateforme de Mise à l'Échelle Automatique et Prédictive	59
7.2 RMSE Globale pour Chaque Valeur de Batch Size	80
7.3 RMSE par TimeSteps Avec la Meilleur Batch Size	80
7.4 RMSE Globale Pour Chaque Valeur d'époque	81
7.5 RMSE par TimeSteps avec le meilleur nombre d'époques	81
7.6 RMSE Globale Pour Chaque Valeur de Batch Size Avec le Meilleur Nombre d'époques Trouvé	82
7.7 RMSE par TimeSteps avec la Meilleur Batch Size avec le Meilleur Nombre d'époque Trouvé	83
7.8 RMSE Globale pour Chaque Taille du Vecteur h	83
7.9 RMSE par TimeSteps avec la meilleure taille du vecteur h	84
7.10 RMSE Globale pour Chaque Taille de la Fenêtre de Prédiction (TimeSteps)	84
7.11 RMSE par TimeSteps avec la Meilleur Taille de la Fenêtre de Prédiction	85
7.12 RMSE Globale en Changeant la Valeur du taux d'apprentissage (Learning Rate α)	86
7.13 Comparaison de la consommation de CPU en utilisant l'approche de mise à l'échelle automatique et prédictive et l'approche de mise à l'échelle automatique non prédictive	86
7.14 Démonstration de MEAP en Utilisant "Grafana"	87
7.15 Maximum consommation de CPU en variant la valeur du seuil de mise à l'échelle	88
B.1 La Fonction Sigmoïde	100
B.2 La Fonction Tanh	101

Liste des tableaux

5.1 Analyse Comparative entre notre Contribution MEAP et les différents Travaux Connexes Analysés précédemment	49
--------------------------------------------------------------------------------------------------------------------------	----

Liste des abréviations

- API** Interfaces de Programmation d'Application
- ARMA** Auto Regressive Moving Average
- ANN** Artificial Neural Networks
- BDD** Base de Données
- CLI** Command Line Interface
- CNN** Convolution Neural Networks
- CNCF** Cloud Native Computing Foundation
- CPU** Central Processing Unit
- CA** Cluster Autoscaler
- CA-NAP** Cluster Autoscaler-Provisionnement Automatique des Noeuds
- ETCD** /ETC Distributed
- HPA** Mise à l'échelle Horizontale des Pods
- IA** Intelligence Artificielle
- K8s** Kubernetes (K - eight characters - S)
- LSTM** Long Short Term Memory
- LR** Linear Regression
- MP** Machine Physique
- MV** Machine Virtuelle
- MLP** Multi Layer Perceptron
- MEAP** Mise à l'Échelle Automatique et Prédictive
- MLP** MultiLayer Perceptrons
- NAP** Provisionnement Automatique des Noeuds

QoS Quality of Service

RNN Recurrent Neural Networks

RMSE Root Mean Square Error

SE Système d'Exploitation

SLO Service-Level Objective

SLA Accords de Niveau de Service

SGD Descente de Gradient Stochastique

VM Virtual Machine

YAML Yet Another Markup Language

VPA Mise à l'échelle Automatique des Pods Verticaux

CSV Comma Separated Values

Introduction Générale

Contents

1.1	Contexte Général	2
1.2	Problématique et Objectifs	3
1.3	Contributions	4
1.4	Organisation du Travail	6

1.1 Contexte Général

Bien que les données aident les différentes industries, telles que les grands fournisseurs des systèmes logiciels (Google, Amazon et Facebook) à mieux servir les clients existants, à atteindre de nouveaux marchés, à simplifier les opérations grâce à l'optimisation et à monétiser les données analysées grâce à une prise de décision rapide, la croissance explosive des données présente un défi majeur. Il est donc prioritaire de fournir, d'une part, une assurance élevée en terme de métriques de **QoS** telles que les temps de réponse, le débit élevé et la disponibilité des services, d'autre part, une utilisation efficace des ressources telles que (**CPU**, Mémoire, Énergie). Le non-respect de ces mesures de **QoS** et de consommation des ressources, entraîne une diminution du nombre d'utilisateurs et une perte importante des revenus.

Avec cette augmentation importante de la demande de trafic ainsi que les services fortement consommateurs de ressources tels que les services d'apprentissage automatique, le cloud computing est adopté comme un acteur principal pour offrir la flexibilité nécessaire et la qualité de service requise. Le terme cloudification est l'action de déplacer l'infrastructure matérielle dans le cloud en utilisant les technologies de virtualisation. Traditionnellement, pour gérer les **MV** sur le cloud, la virtualisation basée sur un hyperviseur était utilisée. Avec les récents progrès de la technologie de virtualisation, la virtualisation au niveau du Système d'Exploitation (**SE**) a été introduite, connue sous le nom de conteneurisation. En comparant avec la virtualisation basée sur des **MV**, la conteneurisation présente l'avantage principalement en raison d'une meilleure portabilité, d'une infrastructure légère, rapide et isolée pour exécuter des applications ainsi qu'une mise à d'échelle facile.

L'environnement basé sur les conteneurs, connue sous le nom cloud natif, a changé la façon dont les utilisateurs conçoivent le processus de développement, de déploiement et de maintenance des applications logicielles. Le nom cloud natif provient des capacités d'isolation natives des conteneurs des systèmes d'exploitation modernes, tout en obtenant une grande flexibilité dans leur déploiement et une faible surcharge en consommation de ressources. Cela a donné naissance à des architectures d'applications basées sur des microservices. Qui consiste à découper une application en services distincts et différents et à les emballer dans des conteneurs distincts et inter-communicants. Ensuite, en les déployant dans un cluster de machines virtuelles ou physiques. La croissance des conteneurs avec la complexité de ces applications a donné naissance à l'orchestration des conteneurs, qui consiste à fournir un mécanisme pour automatiser le déploiement et la configuration, l'auto-réparation et la tolérance aux pannes ainsi que la mise à l'échelle automatique des ressources en fonction des besoins du déploiement. L'un des systèmes d'orchestration les plus utilisés est Kubernetes.

La mise à l'échelle des conteneurs consiste à rajouter des ressources (**CPU**, Mémoire) au même ensemble de conteneurs, faisant référence à la mise à l'échelle verticale. En revanche, l'ajout d'un ensemble de conteneurs à un déploiement existant correspond à la mise à l'échelle horizontale. Ces méthodes de la mise à l'échelle sont utilisées pour maintenir une bonne latence de réponse pour les requêtes simultanées concurrentes, améliorer l'utilisation des ressources (**CPU**, Mémoire) et aider à garantir les Service-Level Objective (**SLO**)s. La plupart des méthodes de mise à l'échelle proposées dans la littérature sont réactives, qui consistent à augmenter les ressources des microservices (**CPU**, Mémoire) pour le cas de la mise à l'échelle verticale et le nombre de conteneurs pour le cas de la mise à l'échelle horizontale, chaque fois que la moyenne **CPU**, mémoire ou l'utilisation réseau franchit un certain seuil [4]. Cependant, la difficulté de trouver des seuils optimaux ainsi que le délai lors de la mise à l'échelle d'un conteneur surchargé, rendent la mise à l'échelle réactive inefficace et incapable de garantir le **SLO** requis.

Plusieurs méthodes de mise à l'échelle automatique et prédictive ont été proposées dans la littérature : [4], [2], [3], [64], leurs idées principales est d'utiliser des techniques d'apprentissage automatique pour prédire les ressources nécessaires pour servir la charge de travail. Ensuite, allouer proactivement des ressources ou augmenter le nombre de conteneurs avant d'atteindre un certain seuil. Leurs objectifs est de minimiser la violation de temps de réponse correspondant aux **SLO**s. Cependant, la plupart des méthodes proposées présentent deux inconvénients : i) la nécessité d'un temps considérable pour faire fonctionner l'application prédictive sur l'infrastructure pour recueillir des données d'entraînements couvrant différents comportements de charge de travail, ii) l'absence de modèle prédictif efficace, accumulant continuellement les données d'entraînement sans affecter les ressources du cluster, où la plupart des approches proposées utilisent des méthodes de prédictions basées sur des modèles de régression tels que : Linear Regression (**LR**), Auto Regressive Moving Average (**ARMA**).

Dans ce qui suit, nous présentons la problématique et objectifs dans la Section 1.2. Ensuite, nous présentons la liste des contributions dans la Section 1.3, et finalement l'organisation de ce travail sera présentée dans la Section 1.4.

1.2 Problématique et Objectifs

Ce travail traite des défis de recherche associés à la mise à l'échelle automatique et prédictive. L'approche du cloud natif doit être mise en œuvre pour tirer pleinement parti de l'approche du cloud computing. Ainsi, la gestion de l'infrastructure d'orchestration doit être traitée avec le problème d'allocation des ressources dans un environnement cloud. En particulier, les questions suivantes doivent être abordées :

- Comment activer la conteneurisation tout en prenant en compte la surveillance des ressources et du trafic ?
- Quelle est la méthode de prédiction la plus adaptée au comportement aléatoire des ressources surveillées ?
- Comment déclencher la mise à l'échelle automatique et prédictive pour améliorer l'utilisation des ressources et éviter le surcharge des microservices ?
- Comment faire apprendre le modèle en continu sans affecter les performances du système de mise à l'échelle ainsi que du cluster ?
- Comment faire une architecture modulaire afin de permettre l'extensibilité du système de mise à l'échelle avec de nouveaux modèles et méthodes prédictives ?

A partir de ces problèmes, nous considérons les objectifs principaux pour ce travail :

- Il est primordiale de bien comprendre les principales technologies permettant de mettre en place le système de mise à l'échelle automatique et prédictive, notamment, le système de conteneurisation Docker le système d'orchestration Kubernetes, la méthode prédiction **LSTM**, le langage de programmation Python et le système de surveillance des données influxdb et Grafana. De ce fait, le premier objectif de ce travail est : Élaboration d'une étude analytique et technique sur les différentes technologies permettant de réaliser l'approche de mise à l'échelle automatique et prédictive.
- Élaboration d'une étude bibliographique comparative sur les différentes approches de mise à l'échelle automatique et prédictive proposée dans la littérature, afin de renforcer le niveau de compréhension, d'améliorer ce qui a été fait, et d'élaborer une approche différente.
- Proposition de l'approche de mise à l'échelle automatique et prédictive **MEAP**, en se basant sur une architecture modulaire et extensible, ensuite, un modèle mathématique pour valider le concept de l'approche ainsi qu'une heuristique pour automatiser le processus de mise à l'échelle.
- Développement du système de mise à l'échelle automatique et prédictive **MEAP**, en prenant comme objectif la minimisation de la consommation des ressources et l'adapter sur l'apprentissage continu ainsi que la modularité.
- La démonstration d'un scénario de mise à l'échelle automatique et prédictive **MEAP** par la visualisation de l'évolution des différents composants de la plateforme.

1.3 Contributions

Dans ce travail, nous proposons une nouvelle méthode de mise à l'échelle automatique et prédictive **MEAP**, pour une application basée sur des microservices hébergée dans une infrastructure

cloud natif Kubernetes K8s, avec l'utilisation du méthode de prédiction **LSTM** qui a montré dans plusieurs travaux de recherches, son efficacité de prédiction et adaptabilité avec des modèles de trafic non réguliers. L'objectif principale est de minimiser la consommation des ressources afin de garantir la **QoS**. Le développement et déploiement du système **MEAP** proposé consiste en : 1) déploiement d'un cluster d'orchestration Kubernetes sur deux machines virtuelles, 2) déploiement des différents composant permettant d'activer la surveillance des ressources (**CPU**, Mémoire), en utilisant le système Metrics-Server de Kubernetes, 3) développement d'un système de stress et de monitoring permettant d'interagir avec le cluster Kubernetes pour stresser le **CPU** d'un microservice puis le monitoring des ressources, 4) développement du modèle prédictif basé sur **LSTM** qui s'entraîne d'une manière continue, sachant que notre travail sera basé sur un ordinateur portable de la marque HP avec un processeur Intel Core i7 et 8 Go de RAM.

Les principaux contributions de ce travail incluent :

- Le choix de la méthode de prédiction **LSTM**, qui pour le meilleur de nos connaissances, nous sommes les premiers à utiliser le type de **LSTM** Encodeur-Décodeur pour un système de mise à l'échelle automatique et prédictive **MEAP**.
- La modularité de notre architecture proposée, dont la majorité des travaux proposés dans la littérature sont dépendant des technologies d'orchestration et difficile à ajouter des extensions ou à changer la méthode de prédiction.
- L'entraînement continu du système prédictif sans affecter les performances du système de mise à l'échelle automatique et prédictive **MEAP** ainsi que le cluster d'orchestration.

Notes bien que, dans ce travail, nous avons considéré que la mise à l'échelle automatique et horizontale, dans lequel on rajoute de nouveaux réplicas ou conteneurs si la consommation des ressources dépasse un certain seuil. En ce qui concerne les ressources nous avons considéré que l'utilisation du **CPU**.

1.4 Organisation du Travail

Ce travail est organisée en 6 chapitres :

Chapitre 1 présente une introduction générale, dans lequel on commence par le contexte général du travail, ensuite la problématique et nos éventuels objectifs, les principaux contributions en comparant avec l'état de l'art, et finalement se termine par l'organisation du rapport.

Chapitre 2, 3 et 4 tout d'abord, donne un aperçu sur le système de conteneurisation Docker, ensuite le système d'orchestration Kubernetes, et finalement la méthode de prédiction **LSTM**.

Chapitre 5 présente l'état de l'art, dans lequel nous soulignons et revoyons la problématique, identifions les lacunes et proposons nos contributions.

Chapitre 6 présente : i) l'architecture globale de notre approche proposée "mise à l'échelle automatique et prédictive" **MEAP**, ii) le modèle prédictif **LSTM** et iii) une modélisation mathématique du problème avec l'heuristique **MEAP**.

Chapitre 7 présent le chapitre tests et résultats, dans lequel, on montre techniquement la plateforme de mise à l'échelle automatique et prédictive ainsi que ses différents composants, le scénario d'exécution ainsi qu'un ensemble de résultats sur l'entraînement et les performances du système **MEAP**.

Chapitre 8 conclut ce travail et synthétise ces objectifs ainsi que les contributions associées et présente une perspective pour des travaux futurs.

Chapitre 2

La technologie de conteneurisation Docker

Contents

2.1	Introduction	8
2.2	Cloud Natif	8
2.3	Les MicroServices	8
2.3.1	Les Avantages des MicroServices	8
2.4	La Virtualisation	8
2.5	La Conteneurisation (Virtualisation par Conteneurs)	9
2.5.1	Ce qui Différencie la Virtualisation de la Conteneurisation	9
2.5.2	Les MicroServices et les Conteneurs	9
2.6	La Technologie des Conteneurs Docker	10
2.6.1	C'est Quoi Docker	10
2.6.2	Architecture et Principaux Composants de Docker	10
2.7	Conclusion	12

2.1 Introduction

Dans ce chapitre nous détaillons un certain nombre de définitions, de concepts de base liées à nos travaux afin de mieux comprendre l'étendu de notre problématique. Nous commençons par présenter les microservices dans 2.3, la virtualisation dans 2.4, les conteneurs dans 2.5 et nous terminons la technologie Docker dans la Section 2.6.

2.2 Cloud Natif

Le cloud natif concentre sur la façon de créer et d'exécuter des applications pour tirer parti du modèle de cloud computing. Les technologies cloud natives telles que Les conteneurs, les orchestrateurs, les maillages de services, les microservices, l'infrastructure immuable et les Interfaces de Programmation d'Application (API) déclaratives permettent aux organisations de créer et d'exécuter des applications évolutives dans des environnements modernes et dynamiques [56].

2.3 Les MicroServices

Contrairement à l'approche monolithique traditionnelle où tous les composants forment une entité indissociable, les microservices fonctionnent en synergie pour accomplir les mêmes tâches, tout en étant séparés. Les microservices désignent une approche de développement logiciel qui consiste à décomposer les applications en éléments les plus simples, indépendants les uns des autres. Chacun de ces composants ou processus est un microservice, granulaire et léger [57].

2.3.1 Les Avantages des MicroServices

- Beaucoup plus faciles à créer, tester, déployer et mettre à jour par rapport aux applications monolithiques ;
- Ils permettent aux organisations de réagir plus rapidement aux nouvelles demandes et d'éviter un processus de développement interminable sur plusieurs années ;
- Les différentes tâches de développement peuvent être réalisées simultanément et de façon agile pour apporter immédiatement de la valeur aux clients [57].

2.4 La Virtualisation

La virtualisation est utilisée pour générer un système physique simulé sur un système physique réel. Elle permet d'utiliser une ressource informatique virtuelle à partir d'une Machine Physique (MP) réelle. Nous pouvons avoir plusieurs systèmes virtuels, appelés MV, fonctionnant sur un

seul système physique. Ces systèmes virtuels partagent l'utilisation des ressources physiques tels qu'un processeur, une interface réseau ou un disque dur, ces derniers sont alloués à une **MV** pour que celle-ci fonctionne comme une **MP** [79].

2.5 La Conteneurisation (Virtualisation par Conteneurs)

Il s'agit d'une forme de virtualisation du **SE** dans laquelle on peut exécuter des applications dans des espaces utilisateurs isolés appelés conteneurs qui utilisent le même **SE** partagé. Un conteneur d'applications est un environnement informatique entièrement regroupé en package et portable et dispose de tout ce qu'une application a besoin pour s'exécuter [57].

Avec les méthodes traditionnelles, coder dans un environnement informatique spécifique entraîne souvent des erreurs et des bogues lorsqu'on transfère ce code dans un nouvel emplacement, la conteneurisation supprime ce problème et permettant de regrouper le code de la demande avec les fichiers de configuration, les dépendances et les bibliothèques associées. On peut ensuite, isoler ce package unique (conteneur) du **SE** hôte, ce qui le rend autonome et portable, c'est-à-dire qu'il peut s'exécuter sur n'importe quelle plate-forme ou n'importe quel cloud sans aucun problème. Cette solution permet aux développeurs de logiciels de créer et de déployer des applications de façon plus rapide et plus sécurisée [1].

2.5.1 Ce qui Différencie la Virtualisation de la Conteneurisation

Dans le cadre de la conteneurisation, le conteneur fait directement appel au **SE** de sa **MP** hôte pour exécuter ses applications. Les conteneurs, partagent le noyau de l'hôte à partir duquel ils sont déployés. Ainsi, il n'est pas nécessaire d'avoir recours à un hyperviseur pour gérer l'ensemble des **MV** qui s'exécuteront via les conteneurs déployés. Parmi les technologies de conteneurisation, on retrouve Docker, comme détaillé dans la Section 2.6.

2.5.2 Les MicroServices et les Conteneurs

Avec les conteneurs, les applications basées sur des microservices disposent d'une unité de déploiement et d'un environnement d'exécution parfaitement adaptés. Lorsque les microservices sont stockés dans des conteneurs, il est plus simple de tirer parti du matériel et d'orchestrer les services, notamment les services de stockage, de réseau et de sécurité. C'est pour cette raison que les microservices et les conteneurs constituent la base du développement d'applications cloud-native [57].

2.6 La Technologie des Conteneurs Docker

2.6.1 C'est Quoi Docker

Docker est un outil open source, développé initialement par Solomon Hykes en 2008 (dotCloud), [14], il est destiné aux développeurs et administrateurs systèmes, dont l'objectif est de faciliter le développement, la diffusion et le déploiement d'applications web autonomes, il représente la plateforme de conteneurisation la plus utilisée, il permet aussi de créer facilement des conteneurs et des applications basées sur les conteneurs, son principal intérêt est d'assembler les briques d'une application en conteneurs pouvant être partagés, sous forme d'images, et de les exécuter quels que soient la plateforme et l'environnement [15].

2.6.2 Architecture et Principaux Composants de Docker

La figure 2.1 représente l'architecture Docker, elle fournit des détails sur les principaux composants d'une plate-forme Docker et sur la manière dont l'utilisateur commande le démon Docker via le client.

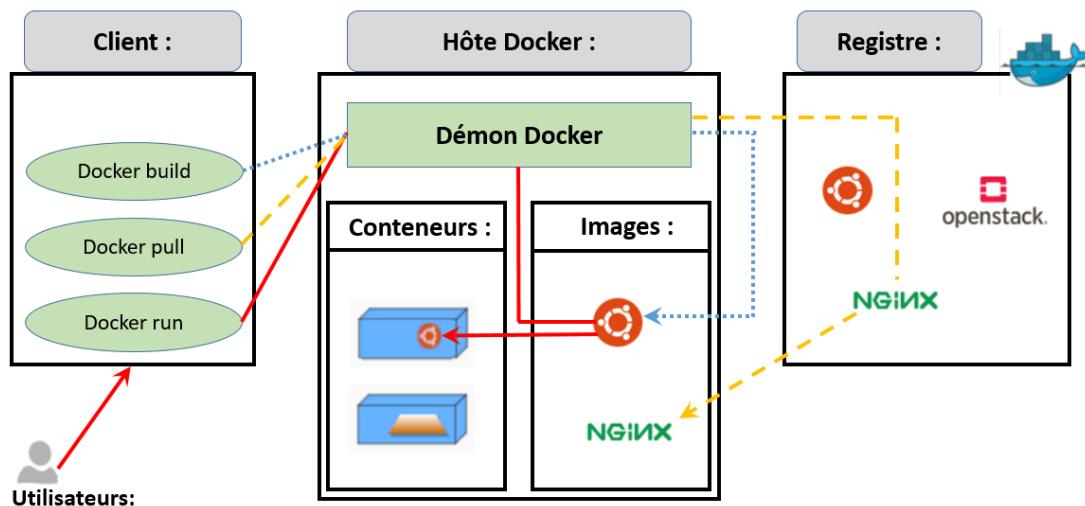


FIGURE 2.1 – Architecture de Docker [Basé sur [8]]

Docker utilise une architecture serveur-client. Où le client est Docker Client (Command Line Interface ([CLI](#))) et le processus serveur est Docker Daemon qui s'exécute sur l'hôte. Le serveur et le client peuvent être sur le même ordinateur.

Voici les différents composants d'une plate-forme Docker.

Image Docker

C'est un modèle en lecture seule, qui sert à créer des conteneurs Docker. Une image Docker est composée de plusieurs couches empaquetant toutes les installations, dépendances, bibliothèques,



FIGURE 2.2 – Image Crée en Utilisant Dockerfile [Basé sur [35]]

processus et codes d’application nécessaires pour un environnement de conteneur pleinement opérationnel.

Pour construire une image, on doit créer un fichier nommé Dockerfile, comme montre la figure 2.2, qui contient une liste d’instructions. Chaque instruction d’un fichier Dockerfile crée une couche dans l’image, lorsqu’on modifie le fichier Dockerfile et reconstruit l’image, seuls les couches qui ont changé sont reconstruits. Cela rend les images petites et rapides par rapport aux autres technologies de virtualisation [25] [24] [66].

Conteneur Docker

Un conteneur Docker est une instance d’image Docker exécutée. On peut gérer, créer, démarrer, arrêter, déplacer ou supprimer un conteneur à l’aide de l’**API** ou de la **CLI** Docker. L’image Docker crée un conteneur Docker.

Par défaut, un conteneur est relativement bien isolé des autres conteneurs et de sa machine hôte, on peut contrôler le degré d’isolement du réseau, du stockage ou d’autres sous-systèmes sous-jacents d’un conteneur par rapport aux autres conteneurs ou à la machine hôte, ce qui permet à l’application d’être exécutée d’une manière isolée [25] [24] [66].

Le client Docker

Le client Docker est le principal moyen utilisé par de nombreux utilisateurs Docker pour interagir avec Docker, le client envoie ces commandes au démon Docker, qui les exécute, on utilisant l’**API** Docker. Ce dernier peut communiquer avec plusieurs démons [24].

Démon Docker

Le démon docker s’exécute en arrière-plan sur le système hôte et sert à contrôler le moteur Docker de manière centralisée. Dans cette fonction, il écoute les requêtes **API** et il crée et gère toutes les images, conteneurs ou réseaux [20].

Registre Docker

Un registre Docker stocke les images Docker. Ce système de catalogage permettant l'hébergement et le "push and pull" des images Docker, on peut même exécuter un registre privé, lorsqu'on utilise les commandes "docker pull" ou "docker run", les images requises sont extraites du registre configuré et lorsqu'on utilise la commande "docker push", l'image est poussée vers le registre configuré.

Docker Hub représente le registre officiel de Docker, il est un registre public que tout le monde peut utiliser [25] [24] [66].

2.7 Conclusion

Dans ce chapitre, nous avons présenté une vue globale sur l'un des systèmes et technologies permettant de concevoir et réaliser notre approche MEAP, qui est le système de conteneurisation Docker.

Dans le chapitre suivant, on va présenter le système d'orchestration Kubernetes.

Chapitre 3

L'orchestration des conteneurs Kubernetes

Contents

3.1	Introduction	14
3.2	L'Orchestration des Conteneurs	14
3.3	Le Système d'orchestration Kubernetes	14
3.3.1	Architecture de Kubernetes	15
3.3.2	Composants d'un Cluster Kubernetes	15
3.3.3	Élément Nécessaire Pour le Déploiement d'un Microservice sur Kubernetes	17
3.4	La Mise à l'Échelle Automatique de Kubernetes (Autoscaling)	19
3.4.1	Le Type HPA	19
3.4.2	Le Type VPA	20
3.4.3	Le type de Mise à l'échelle Automatique du Cluster	20
3.5	Conclusion	21

3.1 Introduction



ans ce chapitre nous détaillons un certain nombre de définitions, de concepts de base liées à la technologie d'orchestration Kubernetes.

3.2 L'Orchestration des Conteneurs

L'orchestration de conteneurs automatise le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs. Les entreprises qui doivent déployer et gérer des centaines ou des milliers de conteneurs et d'hôtes peuvent bénéficier de l'orchestration de conteneurs [62].

L'orchestration offre une visibilité et un contrôle sur l'endroit où les conteneurs sont déployés et permet de déployer différents composants, d'associer dynamiquement des volumes de stockage de données à différents services ou de gérer les aspects de mise en réseau entre ces différents composants. Il permet la gestion automatique des charges de travail sur plusieurs nœuds de calcul (un nœud est tout système connecté à un réseau). Par exemple, ayant six serveurs, mais qu'un serveur démarre un cycle de maintenance, l'orchestrateur peut automatiquement détourner la charge de travail vers les cinq serveurs restants et équilibrer la charge de travail en fonction de ce que les nœuds restants peuvent gérer [63].

3.3 Le Système d'orchestration Kubernetes

Kubernetes est un système Open Source qui exécute et coordonne des applications conteneurisées dans des clusters. Il gère le cycle complet de vie des applications et des services permettant ainsi de limiter le nombre de processus nécessaire au déploiement et à la mise à l'échelle des applications conteneurisées [41], qu'elles soient d'ancienne génération, conteneurisées ou cloud-native, transformées en microservices dans différents environnements [42].

Kubernetes est disponible sur site ou sur le Cloud. Aujourd'hui, **K8s** est maintenu par la Cloud Native Computing Foundation (**CNCF**) de la Linux Foundation. Il s'agit de l'un des principaux outils de développement à l'ère du Cloud Computing [37].

Le développeur n'a plus à s'occuper de la gestion des **MV**, il n'a pas besoin de savoir où sont les applications et l'infrastructure sous-jacente étant masqué pour lui, c'est **K8s** qui va s'occuper de tous ces détails [32].

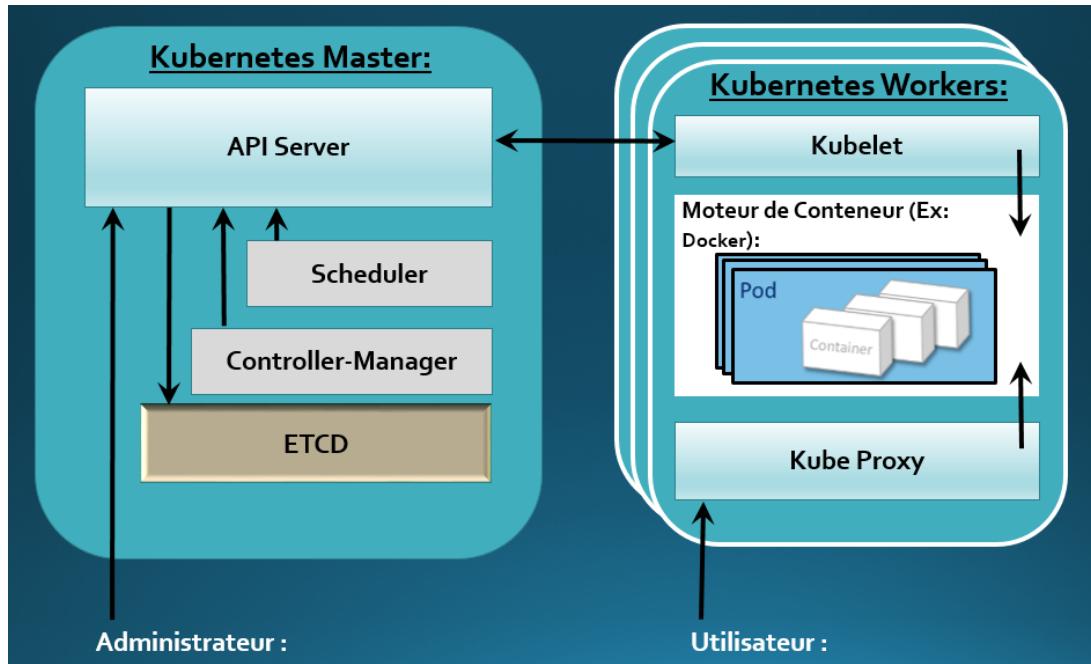


FIGURE 3.1 – Architecture de Kubernetes [Basé sur[9]]

3.3.1 Architecture de Kubernetes

Les déploiements d'applications utilisent souvent plusieurs instances de conteneur appelé clusters. **K8s** permet l'orchestration et la gestion de ces derniers à grande échelle. Un cluster **K8s** se compose d'un ensemble de machines de travail appelées nœuds, qui exécutent des applications conteneurisées. Chacun a au moins un noeud worker. Si on exécute **K8s**, on exécute un cluster. Le plan de contrôle est responsable du maintien du cluster dans un état souhaité, par exemple il vérifie les applications exécutées et les images de conteneurs utilisées. Ainsi, le rôle de **K8s** est de gérer de manière automatisée les besoins en sécurité, données, stockage et mise en réseau des conteneurs [17].

3.3.2 Composants d'un Cluster Kubernetes

Selon la figure 3.1, un cluster **K8s** comprend les principaux composants suivants :

Le nœud maître (Master) et le nœud de travail (Worker)

Le nœud maître exécute le plan de contrôle **K8s** qui est responsable de la gestion des workers, il prend les décisions de planification et met en œuvre des modifications pour conduire le cluster à l'état souhaité. Les workers sont chargés de gérer les applications via des Pods. Chaque nœud a son propre environnement, il peut s'agir d'une **MP** ou **MV**. Chaque nœud exécute des Pods, constitués de conteneurs [60] [41].

L'ETC Distributed (**ETCD**)

C'est est une Base de Données (**BDD**) distribuée et résistante aux pannes de type clé-valeur. **ETCD** stocke et réplique tous les états des clusters et les informations nécessaires au fonctionnement d'un cluster, c'est-à-dire de tous ses composants : les noeuds, les Pods, les configurations, les secrets, les rôles, les comptes [60] [29] [41].

L'API-Server

L'API-Server est la partie "front-end" du plan de contrôle **K8s**. L'API-Server détermine si une requête est valide pour ensuite la traiter. L'API-Server est l'interface utilisée pour gérer, créer et configurer les clusters **K8s**, il garantit la communication entre les utilisateurs, les composants externes et certaines parties du cluster [7].

Le Scheduler

Le scheduler est le planificateur **K8s**, prend en compte les besoins en ressources (par exemple, processeur ou mémoire) d'un Pod, il planifie ensuite l'attribution du Pod au noeud de calcul adéquat. Le planificateur **K8s** assure que le cluster en bonne santé, son intégrité, la possibilité d'intégrer de nouveaux conteneurs si besoin [60].

Le Controller Manager

Il correspond au gestionnaire de contrôleur **K8s**, qui regroupe plusieurs fonctions. Un contrôleur se réfère au planificateur pour assurer qu'un nombre suffisant de Pods est exécuté, si un Pod est défaillant, le contrôleur le remarque et réagit. Il connecte les services aux Pods afin que les demandes soient acheminées jusqu'aux points de terminaison appropriés [60].

Le Pod

Le Pod est l'unité la plus petite et la plus simple dans le modèle d'objets de **K8s**. Il représente une instance unique d'une application. Comme montre la figure 3.2, chaque Pod est constitué d'un conteneur ou d'une série de conteneurs étroitement couplés, ainsi que des options permettant de contrôler l'exécution de ces conteneurs, chaque Pod au sein d'un node possède : 1) Une adresse IP locale, 2) Un ou plusieurs conteneurs, 3) Un ou plusieurs volumes associés à ces conteneurs, c'est-à-dire des ressources de stockage persistant.

Le Service

Un service maintient une liste logique de Pods qui acceptent le trafic entrant et exposent un port interne pour accéder aux Pods sous-jacents, il gère les changements de trafic en modifiant le nombre de Pods [61] [41].

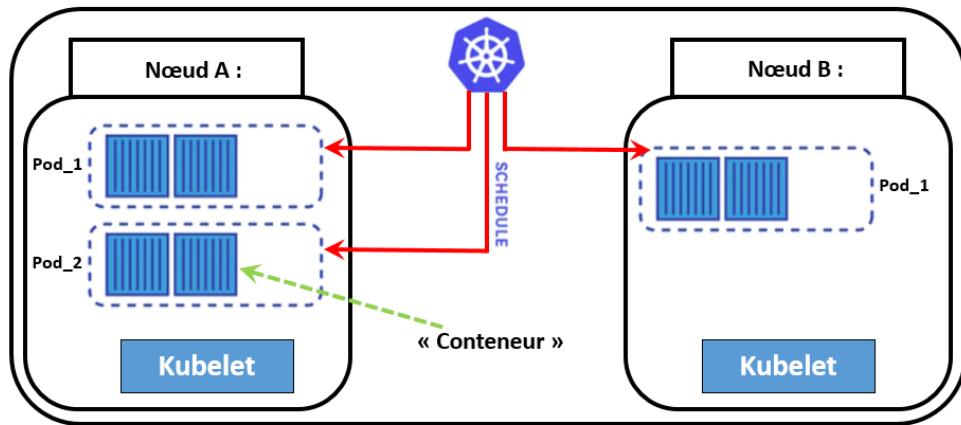


FIGURE 3.2 – Architecture d'un Pod [Basé sur [67]]

Le Composant kubelet

L'agent de surveillance de Kubernetes (kubelet) est une minuscule application située au sein de chaque nœud de travail qui communique avec le plan de contrôle. Le kubelet s'assure que les conteneurs sont exécutés dans un Pod. Lorsque le plan de contrôle envoie une requête vers un nœud, le kubelet exécute l'action. Si un noeud ne répond plus, les Pods déployés sur ce noeud de travail seront transférés à un autre noeud de travail pour assurer la stabilité [26] [40].

Le Composant kube-proxy

Chaque nœud de calcul contient également un proxy réseau appelé « kube-proxy » qui facilite la mise en œuvre des services de mise en réseau de K8s et gère les communications réseau dans et en dehors du cluster [60].

Moteur d'Exécution de Conteneurs (Container Runtime)

Chaque nœud de calcul dispose d'un moteur qui permet d'exécuter les conteneurs, Docker en est un exemple [60].

3.3.3 Élément Nécessaire Pour le Déploiement d'un Microservice sur Kubernetes

Dans cette section, on va présenter quelques éléments nécessaires pour le déploiement d'une application sur Kubernetes :

- **kubectl**

Interface en ligne de commande dans laquelle on peut gérer votre cluster K8s [60].

- **Volume**

Un volume Kubernetes est un répertoire qui contient des données accessibles aux conte-

neurs d'un Pod donné dans la plate-forme d'orchestration et de planification. Les volumes fournissent un mécanisme de plug-in pour connecter des conteneurs éphémères à des données persistants ailleurs.

- **L'Espace de Nom (NameSpace)**

Les espaces de noms fournissent un mécanisme pour isoler des groupes de ressources au sein d'un seul cluster **K8s**. Les noms de ressources doivent être uniques au sein d'un même namespace, mais pas entre namespaces [41].

- **Le Composant ReplicaSet**

Un ReplicaSet est conçu pour maintenir un ensemble stable de Pods à tout moment.

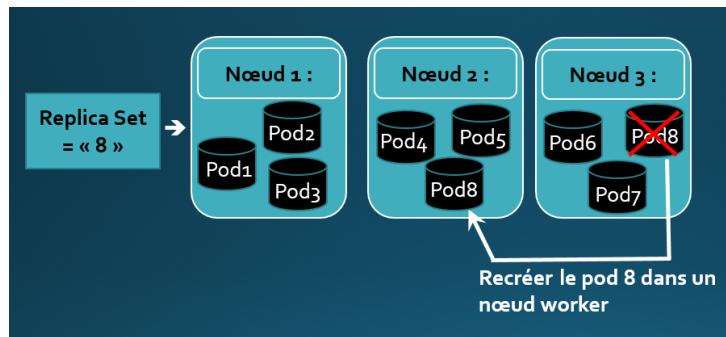


FIGURE 3.3 – Architecture de Réplicaset [Basé sur [22]]

Il garantit la disponibilité d'un certain nombre de Pods. Un ReplicaSet atteindra son objectif en créant et en supprimant des Pods pour atteindre le nombre de réplicas souhaité. Lorsqu'un ReplicaSet doit créer un nouveau Pod, il utilise alors son Pod template [68].

- **Déploiement**

Leur but est de maintenir un ensemble de Pods identiques en cours d'exécution et de les mettre à niveau de manière contrôlée - en effectuant une mise à jour continue par défaut. Ils permettent de déployer une version spécifique de l'application et de spécifier le nombre de Pods dont on a besoin pour qu'elle soit opérationnelle [61] [21].

- **StatefulSets**

StatefulSet est l'objet de l'**API** de charge de travail qui gère le déploiement et la mise à l'échelle d'un ensemble de Pods basés sur une même spécification de conteneur, et fournit des garanties sur l'ordre et l'unicité de ces Pods. Contrairement à un Déploiement, un StatefulSet maintient une identité pour chacun de ces Pods [72].

- **Registre de Conteneurs**

Le registre de conteneurs stocke les images de conteneurs sur lesquelles repose **K8s**. Il peut s'agir d'un registre tiers ou d'un registre qu'on a configuré [60].

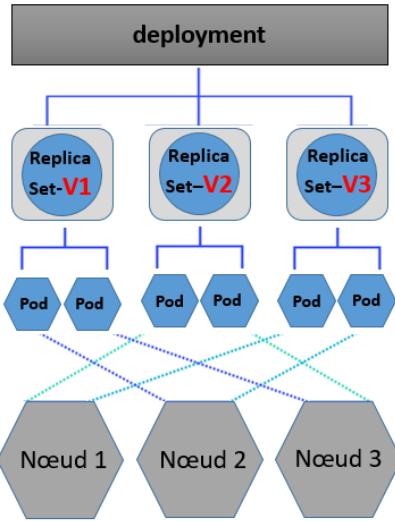


FIGURE 3.4 – Déploiement [Basé sur [22]]

3.4 La Mise à l’Échelle Automatique de Kubernetes (Autoscaling)

K8s offre plusieurs niveaux de contrôle de la gestion de la capacité pour la mise à l'échelle automatique. Les planificateurs K8s attribuent des Pods de conteneurs aux nœuds de cluster avec l'ensemble du processus contrôlable par les paramètres de configuration dans les fichiers Yet Another Markup Language (YAML). À l'aide de ces fichiers, les administrateurs K8s peuvent demander et définir des limites maximales pour le processeur et la mémoire disponibles pour une utilisation par chaque conteneur dans un Pod [43].

Les administrateurs peuvent également fournir des instructions à K8s pour allouer automatiquement plus de CPU et de mémoire à un Pod en fonction des critères d'utilisation du CPU et de la mémoire [également appelé *mise à l'échelle automatique des Pods verticaux ou bien VPA en anglais*]. En outre, ils peuvent configurer K8s pour répliquer automatiquement les Pods pour les charges de travail d'application sans état [également appelée *mise à l'échelle automatique des Pods horizontaux ou bien HPA en anglais*]. Enfin, ils peuvent également configurer le cluster pour ajouter plus de nœuds une fois que les autres nœuds sont entièrement utilisés ou réservés [également appelé *Mise à l'échelle automatique du cluster*] [43].

3.4.1 Le Type HPA

Ajoute et supprime des Pods. Une fonctionnalité K8s pour augmenter ou diminuer le nombre de réplicas de Pod automatiquement en fonction de métriques définies [77].

3.4.2 Le Type VPA

Augmente et diminue le processeur et la mémoire du Pod, c'est une fonctionnalité K8s pour dimensionner correctement les Pods de déploiement et éviter les problèmes d'utilisation des ressources sur le cluster [77].

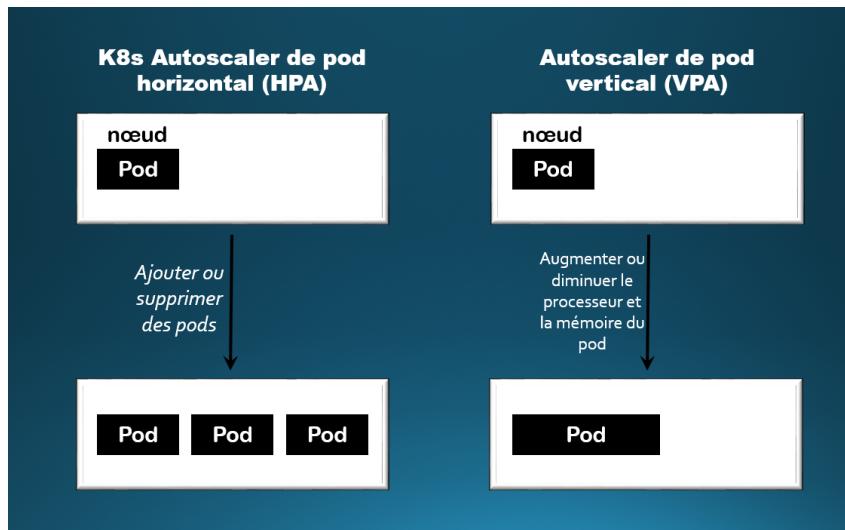


FIGURE 3.5 – HPA VS VPA [Basé sur [77]]

3.4.3 Le type de Mise à l'échelle Automatique du Cluster

Ajoute et supprime des nœuds de cluster. À mesure que de nouveaux Pods sont déployés et que le nombre de réplicas pour les Pods existants augmente, les nœuds de travail du cluster peuvent utiliser toutes leurs ressources allouées. Par conséquent, plus aucun Pod ne peut être programmé sur les nœuds de calcul existants. Certains Pods peuvent entrer dans un état d'attente. Attendre le processeur et la mémoire et éventuellement créer une panne. En tant qu'administrateur K8s, vous pouvez résoudre manuellement ce problème en ajoutant plus de nœuds de travail au cluster pour permettre la planification de Pods supplémentaires. Le problème est que ce processus manuel prend du temps et s'adapte mal. Heureusement, K8s Cluster Autoscaler peut résoudre ce problème en automatisant la gestion de la capacité. Plus précisément, Cluster Autoscaler automatise le processus d'ajout et de suppression de nœuds de travail d'un cluster K8s [77].

3.5 Conclusion

Dans ce chapitre, nous avons présenté une vue globale sur l'un des systèmes et technologies permettant de concevoir et réaliser notre approche **MEAP**, qui est le système d'orchestration Kubernetes.

Dans le chapitre suivant, on va présenter la méthode de prédiction **LSTM**.

La méthode de prédition LSTM

Contents

4.1	Introduction	23
4.2	La Méthode de Prédition (LSTM)	23
4.2.1	L'Apprentissage Automatique (Machine Learning)	23
4.2.2	Apprentissage Profond (Deep Learning)	23
4.2.3	Différents Types de Réseaux de Neurones dans Deep Learning	23
4.2.4	Entraîner LSTM	32
4.2.5	Modes d'Utilisations de LSTM	35
4.2.6	Types des Modèles LSTM	38
4.2.7	Préparation des Données pour les Réseaux LSTM	40
4.3	Conclusion	42

4.1 Introduction



ans ce chapitre nous détaillons un certain nombre de définitions, de concepts de base liées à la la méthode de prédiction **LSTM** .

4.2 La Méthode de Prédiction (**LSTM**)

Dans cette section, on va commencer par donner une vue globale sur l'apprentissage automatique et l'apprentissage profond, comme ils sont la base pour bien comprendre **LSTM**.

4.2.1 L'Apprentissage Automatique (Machine Learning)

L'apprentissage automatique est une technique utilisée dans Intelligence Artificielle (**IA**). Il se compose de modèles formés dans une base de connaissances pour effectuer des tâches complexes. Il s'agit d'une technique de programmation informatique qui utilise la probabilité statistique pour permettre aux ordinateurs d'apprendre par eux-mêmes sans être explicitement programmés. Pour son objectif fondamental, l'apprentissage automatique apprend aux ordinateurs à apprendre, puis à agir et à réagir comme des humains, améliorant de manière autonome leur façon d'apprendre et ce qu'ils savent au fil du temps. Les programmes de développement utilisés par l'apprentissage automatique s'ajustent chaque fois qu'il est exposé à un type différent de données d'entrée [53] [54].

4.2.2 Apprentissage Profond (Deep Learning)

L'apprentissage profond est un type d'**IA** dérivé de l'apprentissage automatique. Il s'appuie sur un réseau de neurones artificiels s'inspirant du cerveau humain. Ce réseau est composé de dizaines, voire de centaines de couches de neurones, chacune recevant et interprétant les informations de la couche précédente. Le système apprendra par exemple à reconnaître les lettres avant de s'attaquer aux mots dans un texte, ou détermine s'il y a un visage sur une photo avant de découvrir de quelle personne il s'agit [19].

4.2.3 Différents Types de Réseaux de Neurones dans Deep Learning

Il y a trois types importants de réseaux de neurones qui constituent la base de la plupart des modèles pré-formés en apprentissage profond [23] :

- Réseaux de neurones artificiels (**ANN**) ;
- Réseaux de neurones convolutifs (**CNN**) ;
- Réseaux de neurones récurrents (**RNN**).

On va se concentrer beaucoup plus sur les **RNN** qui sont la base pour bien comprendre La méthode **LSTM**.

4.2.3.1 Les Réseaux de Neurones Artificiels (**ANN**)

Le réseau de neurones artificiel a la même fonction et nature que les réseaux de notre cerveau. Les données sont transférées dans le neurone par l'entrée, et les données sont envoyées en sortie après traitement. Les **ANNs** aident à effectuer des tâches telles que la classification des données [5].

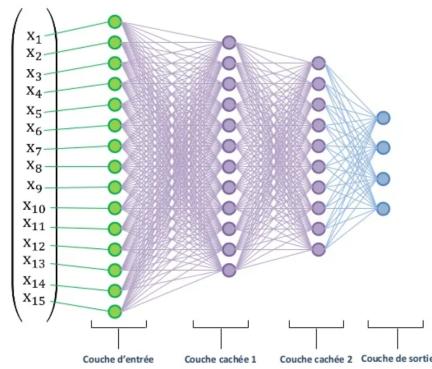


FIGURE 4.1 – Architecture des **ANN** [6]

Parmi les limites des architectures traditionnelles (**ANN**) on peut citer [6] :

- Taille fixe de la couche d'entrée et de la couche de sortie ;
- Architecture fixe du réseau ;
- Ne tiennent pas compte de l'ordre des données ;
- Non adaptées aux séries temporelles .

4.2.3.2 Les Réseaux de Neurones Convolutifs (**CNN**)

Réseaux de neurones convolutifs est un type de réseau de neurones artificiels largement utilisé pour la reconnaissance et la classification d'images/d'objets. Le Deep Learning reconnaît ainsi des objets dans une image en utilisant un **CNN**. Les **CNN** jouent un rôle majeur dans diverses tâches/fonctions telles que les problèmes de traitement d'image, les tâches de vision par ordinateur telles que la localisation et la segmentation, l'analyse vidéo, la reconnaissance des obstacles dans les voitures autonomes, ainsi que la reconnaissance vocale dans le traitement du langage naturel. Comme les **CNN** jouent un rôle important dans ces domaines à croissance rapide et émergents, ils sont très populaires dans le Deep Learning [18].

4.2.3.3 Les Réseaux de Neurones Récursifs (RNN)

Les **RNNs** sont une catégorie de réseaux de neurones dédiée au traitement de séquences.

On peut dire que les réseaux de neurones récurrents reposent sur deux principes : 1) Un **RNN** peut traiter des données de taille variable, 2) Utilisation de connexions récurrentes qui permettent d'analyser la partie passée du signal (se "souvenir" de sa décision à un instant précédent).

Dans les **RNNs** la sortie du réseau d'un pas de temps est fournie comme entrée dans le pas de temps suivant, cela permet au modèle de prendre une décision quant à ce qu'il faut prédire en fonction à la fois de l'entrée pour l'étape de temps actuelle et de la connaissance directe de ce qui a été produit à l'étape de temps précédente.

Dans la méthode traditionnelle, nous n'étions pas en mesure de considérer collectivement les différents intrants et extrants. Même si l'information était connectée, nous la considérions comme un individu. Cela a créé divers défis pour de nombreuses tâches. Par exemple, dans un problème de prédiction d'un mot à partir d'une séquence de mots, il est évident qu'il faut connaître le premier mot pour prédire le mot suivant car les deux sont interconnectés. Le **RNN** traite la même tâche de la même manière, en conservant les données dans un ordre précis et peut revenir sur quelques étapes pour utiliser les informations précédentes pour les résultats actuels [5] [70].

À l'exception de la notion de récursivité, les **RNN** sont similaires aux réseaux de neurones classiques (perceptron multicouche) appelés MultiLayer Perceptrons (**MLP**) (Le **MLP** est un type de **ANN**). Les **MLP** sont des algorithmes d'apprentissage automatique supervisé, prennent la description d'un objet en entrée, et fournissent une prédiction en sortie. L'entrée est représentée par un vecteur numérique, qui décrit les caractéristiques (features) de l'objet. Ce vecteur traverse une succession de couches de neurones, où chaque neurone est une unité de calcul élémentaire. La prédiction est fournie en sortie sous la forme d'un vecteur numérique [58].

Dans le schéma suivant, une couche de **MLP** reçoit en entrée un vecteur, par exemple : $[a, b, c]$, et produit en sortie un vecteur, par exemple $[h_1, h_2, h_3, h_4]$, et nous pouvons représenter une couche entière sous la forme d'une cellule [58].

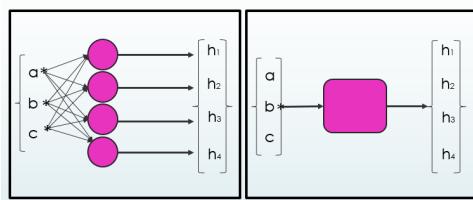


FIGURE 4.2 – Architecture des **MLP** [Basé sur [58]]

4.2.3.4 Limites des RNN Simples

Le plus souvent les **RNNs** ne gèrent pas bien les dépendances à long terme à cause de la disparition du gradient (vanishing gradient) comme conséquence de l'utilisation de "tanh".

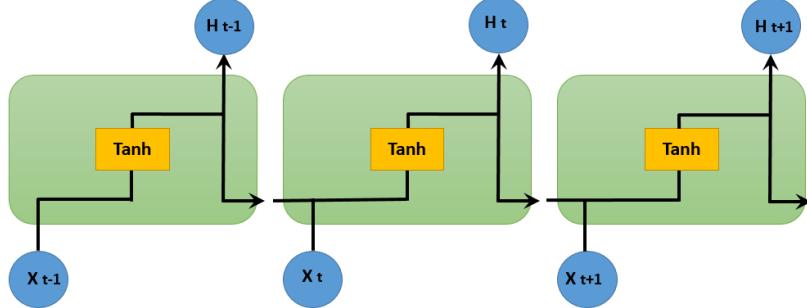


FIGURE 4.3 – Tanh [Basé sur [6]]

Problème de la Disparition du Gradient (Vanishing Gradient)

L'une des principales difficultés pour entraîner les **RNNs** est le problème du disparition du gradient. L'apprentissage d'un réseau de neurones consiste à ajuster les poids " W ", associées aux neurones, afin de minimiser une erreur donnée par une fonction de coût " L ". Cette erreur mesure l'écart entre les labels réels y (la vérité) et les labels prédicts \bar{y} (les données prédictives), elle est mesurée sur une partie des données d'entraînement.

Ainsi, si notre modèle est composé de trois cellules et est représenté par les fonctions : $f.g.h$, son gradient par rapport à W sera $df/dg \times dg/dh \times dh/dW$.

Dans le cas d'une couche RNN, on a :

$$\begin{cases} h_1 = f_w(x_1, h_0) \\ h_2 = f_w(x_2, h_1) = f_w(x_2, f_w(x_1, h_0)) \\ h_3 = f_w(x_3, h_2) = f_w(x_3, f_w(x_2, f_w(x_1, h_0))) \end{cases}$$

Avec $f_w(x, h) = \tanh(W^T \text{concat}(x, h) + b)$ et une fonction de coût $L(h_3, y)$.

Le gradient de h_3 par rapport à W est donc proportionnel à la multiplication de dérivées de la fonction tangente hyperbolique, tel que : la tangente hyperbolique a tendance à écraser les valeurs qu'elle prend en entrée. En effet, elle est définie sur l'espace des réels mais ses valeurs de sortie sont dans l'intervalle $[-1, 1]$. Ainsi, après un premier passage par la tangente hyperbolique, on obtient une valeur entre -1 et 1 . Ensuite, comme $\tanh(1) = 0.76$ et $\tanh(-1) = -0.76$, un deuxième passage par la fonction tanh résulte en un intervalle de valeurs encore plus réduit, et ainsi de suite [51].

Et comme la dérivée de cette fonction se situe dans l'intervalle $[0, 1]$, plus on multiplie des valeurs

entre 0 et 1 entre elles, plus le résultat se rapproche de 0. Le gradient prend donc des valeurs très petites lorsque les séquences sont longues. La mise à jour des paramètres devient donc très lente et l'entraînement du modèle est mis à mal [51].

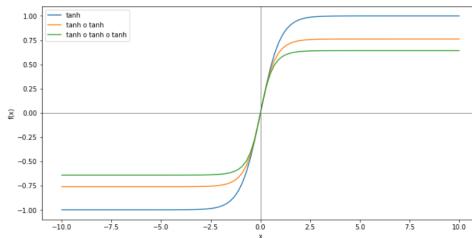


FIGURE 4.4 – Passages Successifs sur la Fonction Tangente Hyperbolique [51]

Problème de Dépendance à Long Terme

Comme expliqué dans la section précédente, le problème de disparition du gradient provoque le problème de dépendance à long terme, c'est à dire, plusieurs dérivations successives qui donnent des valeurs du gradient plus proches de zéro, d'où vient le nom disparition.

Prenons un exemple pour bien comprendre c'est quoi ce problème. Considérons une phrase comme, "Les nuages sont dans le - - - -". Le modèle RNN peut facilement prédire "Sky" ici et cela est dû au contexte des nuages est très proche, cela vient en entrée de la couche précédente. Mais ce n'est peut-être pas toujours le cas [65]. Imaginons si on a une phrase comme : « Jane est née au Kerala. Jane avait l'habitude de jouer pour l'équipe de football féminine et a également remporté les examens de niveau de l'état. Jane parle - - - - couramment ». C'est une très longue phrase et le problème ici est que, en tant qu'humain, on peut dire que, depuis que Jane est née au Kerala et a réussi son examen d'état, il est évident que vous devez maîtriser le "malayalam" très couramment. Mais, Comment notre machine le sait-elle ? Au point où le modèle veut prédire les mots, on peut oublier le contexte du Kerala et plus d'autre chose. C'est le problème de la dépendance à long terme sur RNN [65].

Une façon de résoudre le problème du gradient de fuite et de la dépendance à long terme au RNN est d'opter pour les réseaux LSTM.

4.2.3.5 Long Short Term Memory (LSTM)

Comme nous l'avons vu dans la section précédente, afin de modéliser des dépendances à long terme, il est nécessaire de donner aux réseaux de neurones récurrents la capacité de maintenir un état sur une longue période de temps. C'est le but des cellules LSTM, qui possèdent une mémoire interne appelée cellule. La cellule permet de maintenir un état aussi longtemps que

nécessaire [45]. Les **LSTMs** se composent de plusieurs unités appelées *Cellule*, chaque cellule se compose de trois portes appelées portes d'entrée, sortie et oubli. Les portes d'oubli s'occupent des informations qui doivent être autorisées à traverser le réseau. De cette façon, nous pouvons avoir une mémoire à court et à long terme. La figure 4.5 montre l'architecture de **LSTM** [65]. L'idée de **LSTM** est de diviser le signal entre ce qui est important à court terme à travers l'état

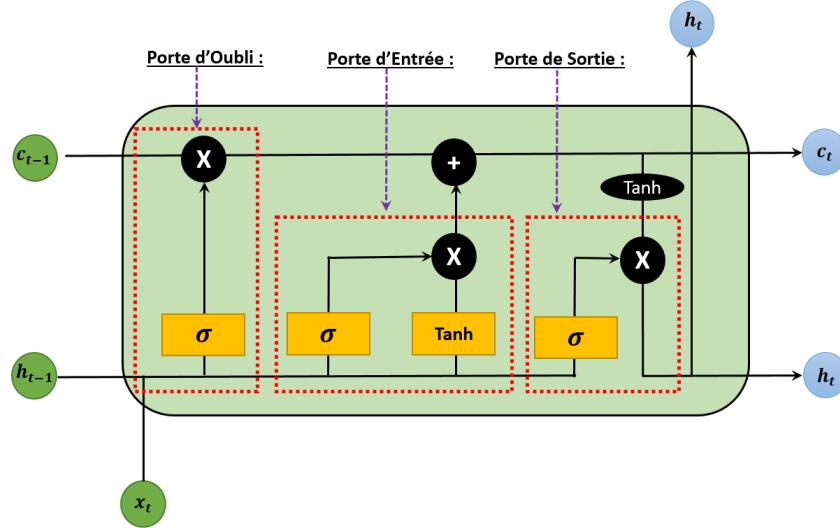


FIGURE 4.5 – **LSTM** Architecture [Basé sur [48]]

caché (hidden state), et ce qui l'est à long terme, à travers la cellule d'état (cell state). Ainsi, le fonctionnement global d'un **LSTM** peut se résumer en trois étapes [51] :

- Déetecter les informations pertinentes venant du passé, piochées dans le cell state à travers la forget gate ;
- Choisir, à partir de l'entrée courante, celles qui seront pertinentes à long terme, via l'input gate. Celles-ci seront ajoutées au cell state qui fait office de mémoire longue ;
- Piocher dans le nouveau cell state les informations importantes à court terme pour générer le hidden state suivant à travers l'output gate.

4.2.3.6 Cellule **LSTM** Pas à Pas

Comme **LSTM** est conçu pour éviter le problème de dépendance à long terme. On introduit une nouvelle variable C_t qui permettra de se souvenir des informations pendant de longues périodes.

La mise à jour des variables se fera alors selon les équations suivante :

$$\begin{cases} C_t = C_{t-1} + \tanh(W_{hg} h_{t-1} + W_{xg} x_t + b_g) \\ h_t = \tanh(C_t) \end{cases} \quad (4.1)$$

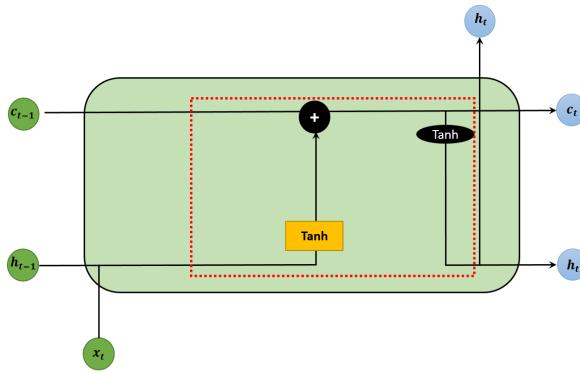


FIGURE 4.6 – Architecture d'une Cellule LSTM [Basé sur [6]]

Cette fois on est assuré que l'état caché h_t calculé à instant t tient compte de tous les états cachés précédent qui sont conservés dont l'état de la cellule C_t .

Cela présente des inconvénients. Dont, le passé est toujours aussi important que le présent ce qui n'est pas pertinent. la solution idéale pour cela est de permettre aux réseaux de garder ou d'oublier les informations du passé selon qu'elles soient déterminantes ou pas [6].

Solution Adoptée - Porte d'Oubli

Comme montre la figure 4.7 la solution adoptée est de pondérer la mémoire du passé représentée par C_t par un réseau de neurones avec une seule couche sigmoïde. Ce réseau va apprendre à oublier les informations non pertinentes. Il est appelé porte d'oubli ou forget gate.

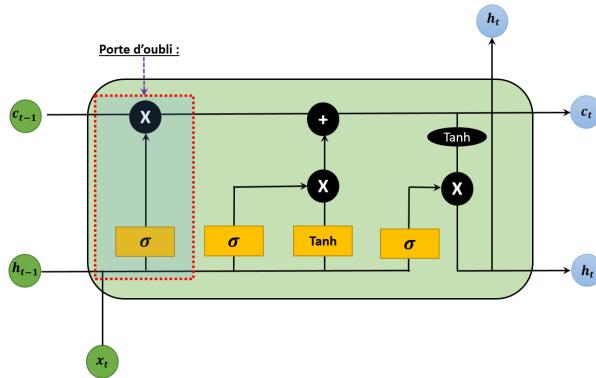


FIGURE 4.7 – Représentation de la porte d'oubli [Basé sur [6]]

Cette porte décide de quelle information doit être conservée ou jetée : l'information de l'état caché précédent est concaténé à la donnée en entrée (par exemple le mot « des » vectorisé) puis on y applique la fonction *sigmode* afin de normaliser les valeurs entre 0 et 1. Si la sortie de la *sigmode* est proche de 0, cela signifie que l'on doit oublier l'information et si on est proche de 1 alors il faut la mémoriser pour la suite [6] [33].

$$\begin{cases} f_t = \sigma(w_{hf} \times h_{t-1} + w_{xf} \times x_t + b_f).O : \\ \sigma(X) = 1/(1 + e^{-1}), \text{ Donc : } 0 \leq \sigma(X) \leq 1 \\ C_t = f_t \times C_{t-1} + \tanh(W_{hg} \times h_{t-1} + W_{xg} \times x_t + b_g) \\ h_t = \tanh(C_t) \end{cases} \quad [6] \quad (4.2)$$

Si par exemple :

$$C_{t-1} = \begin{pmatrix} 0.2 \\ -0.4 \\ 0.75 \\ 0.83 \end{pmatrix} \text{ et } f_t = \begin{pmatrix} 0.9 \\ 0.01 \\ 0.3 \\ 0.95 \end{pmatrix} \text{ Alors : } C_t = \begin{pmatrix} 0.18 \\ 0.00 \Rightarrow \text{Oubli} \\ 0.23 \\ 0.79 \end{pmatrix}$$

Donc, l'oubli est appliquée au deuxième élément de C_{t-1} . le premier et le troisième élément restent pratiquement inchangé donc on dit qu'ils sont conservés [6].

Porte d'Entrée

Une autre porte est ajouté pour pondérer la mise à jour (additive) de l'état de la cellule C_t par la sortie "Tanh" prenant comme entrée h_{t-1} et x_t .

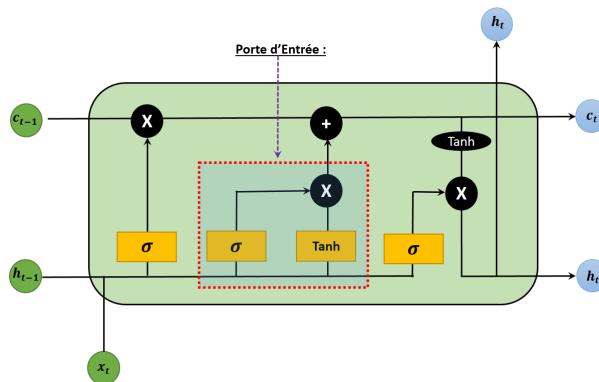


FIGURE 4.8 – Présentation de la Porte d'Entrée [Basé sur [6]]

Cette porte qui s'appelle input gate va apprendre à utiliser, à ignorer ou à moduler les informations entrées selon leur importance [6].

- *Sigmoid* va renvoyer un vecteur pour lequel une coordonnée proche de 0 signifie que la coordonnée en position équivalente dans le vecteur concaténé n'est pas importante.
- A l'inverse, une coordonnée proche de 1 sera jugée importante (i.e. utile pour la prédiction que cherche à faire le **LSTM**).

- *Tanh* va simplement normaliser les valeurs (les écraser) entre -1 et 1 pour éviter les problèmes de surcharge de l'ordinateur en calculs.
- Le produit des deux permettra donc de ne garder que les informations importantes, les autres étant quasiment remplacées par 0 [6] [33].

$$\begin{cases} f_t = \sigma(w_{hf} \times h_{t-1} + w_{xf} \times x_t + b_f). \\ i_t = \sigma(w_{hi} \times h_{t-1} + w_{xi} \times x_t + b_i). \\ C_t = f_t \times C_{t-1} + i_t \times \tanh(W_{hg} \times h_{t-1} + W_{xg} \times x_t + b_g). \\ h_t = \tanh(C_t). \end{cases} \quad [6]. \quad (4.3)$$

L'État de la Cellule

On parle plus de l'état de la cellule avant d'aborder la dernière porte (porte de sortie), car la valeur calculée ici est utilisée dedans.

Dans l'état de la cellule, d'abord on multiplie coordonnée à coordonnée la sortie de l'oubli avec l'ancien état de la cellule. Cela permet d'oublier certaines informations de l'état précédent qui ne servent pas pour la nouvelle prédiction à faire. Ensuite, on additionne le tout (coordonnée à coordonnée) avec la sortie de la porte d'entrée, ce qui permet d'enregistrer dans l'état de la cellule ce que le **LSTM** (parmi les entrées et l'état caché précédent) a jugé pertinent [33].

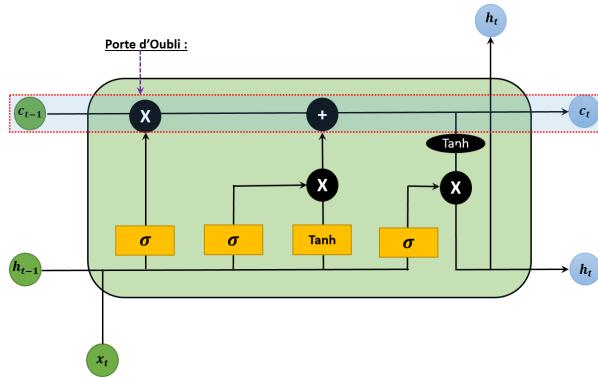


FIGURE 4.9 – Représentation de l'État de la Cellule [Basé sur [6]]

Porte de Sortie

La porte de sortie est ajoutée pour pondérer la mise à jour de l'état caché h_t . Cela permet de décider quelles informations l'état caché h_t doit porter, cette porte est appelé porte de sortie ou

output gate [6].

$$\begin{cases} f_t = \sigma(w_{hf} \times h_{t-1} + w_{xf} \times x_t + b_f). \\ i_t = \sigma(w_{hi} \times h_{t-1} + w_{xi} \times x_t + b_i). \\ o_t = \sigma(w_{ho} \times h_{t-1} + w_{xo} \times x_t + b_o). \\ C_t = f_t \times C_{t-1} + i_t \times \tanh(W_{hg} \times h_{t-1} + W_{xg} \times x_t + b_g). \\ h_t = o_t \times \tanh(C_t). \end{cases} \quad [6]. \quad (4.4)$$

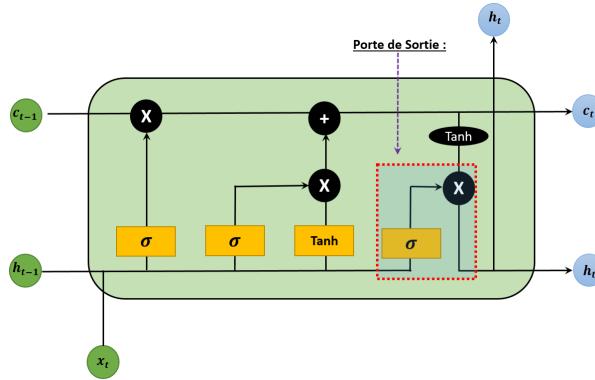


FIGURE 4.10 – Représentation de la Porte de Sortie [Basé sur [6]]

Si par exemple [6] :

$$\tanh(c_t) = \begin{pmatrix} -0.8 \\ -0.3 \\ 0.64 \\ 0.83 \end{pmatrix} \text{ et } \sigma_t = \begin{pmatrix} 0.96 \\ 0.1 \\ 0.01 \\ 0.38 \end{pmatrix} \text{ Alors : } h_t = \begin{pmatrix} -0.77 \Rightarrow \text{Retenu} \\ -0.03 \\ 0.01 \Rightarrow \text{Non Retenu} \\ 0.00 \end{pmatrix}$$

4.2.4 Entrainer LSTM

Comme montré dans des sections précédentes, un réseau de neurones est une séquence de fonctions simples, prenant en entrée les données de notre problème. Cette séquence de fonctions prend comme paramètres des poids "W". L'apprentissage consiste à modifier les poids "W" afin de minimiser une erreur données par une fonction de coût "L". Cette erreur mesure la différence entre les données réels "y" et les données prédites par le modèle " \bar{y} ". L'entraînement se déroule comme suit :

- Les poids W sont initialisés aléatoirement,
- On fait passer X_{train} par le modèle pour obtenir une prédiction \bar{y}_{train} ,
- On calcule ensuite la valeur de la fonction de coût $L(y_{train}, \bar{y}_{train})$,

- On calcule le gradient de cette fonction L par rapport aux paramètres W : (dw_1 et dw_2),
- On cherche à minimiser la fonction de coût en mettant à jour les paramètres W . Pour cela, on effectue une descente de gradient grâce au gradient calculé à l'étape précédente.

$$f(W, b) = \begin{cases} W = W - \alpha \cdot d_W \\ b = b - \alpha \cdot d_b \end{cases} \quad (4.5)$$

4.2.4.1 La Rétro-Propagation du Gradient

Dans les réseaux de neurones, on fait la propagation vers l'avant pour obtenir la sortie d'un modèle et vérifier si cette sortie est correcte, pour obtenir l'erreur, on fait Rétro-propagation (en anglais : Backward-Propagation), qui n'est rien d'autre que de revenir en arrière à travers le réseau de neurones pour trouver les dérivées partielles de l'erreur par rapport aux poids " W ", ce qui nous permet de soustraire les valeurs des poids.

Ensuite, ces dérivées sont utilisées par "descente de gradient", qui est un algorithme utilisé afin de minimiser itérativement une fonction donnée. Ensuite, il ajuste les poids vers le bas ou vers le haut, de façon à diminuer l'erreur [69].

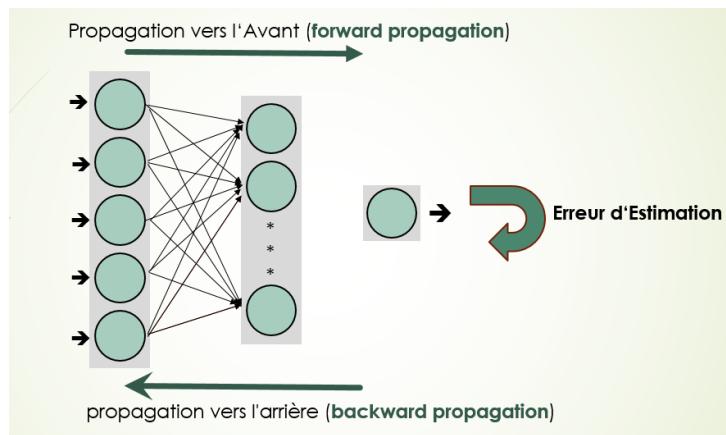


FIGURE 4.11 – Rétro-propagation du gradient [Basé sur [69]]

Exemple Illustratif

Soit x_t l'entrée à l'instant t dans une cellule LSTM, l'état de la cellule à partir de l'instant $t-1$ et t soit c_{t-1} et c_t et la sortie pour l'instant $t-1$ et t soit h_{t-1} et h_t . La valeur initiale de c_t et h_t à $t=0$ sera zéro.

La Propagation vers l'Avant (The Forward Propagation), [27] :

- Étape 1 : Initialisation des Poids

-La Porte d'entrée : $w_{xi}, w_{xg}, w_{hg}, w_{hi}, b_i, b_g$;

-La Porte d'oubli : w_{xf} , b_f , w_{hf} ;

-La Porte de sortie : w_{xo} , b_o , w_{ho} .

- Étape 2 : Passage par Différentes Portes :

Les Entrées : x_t et h_{t-1} , c_{t-1} sont données à la cellule LSTM.

Passage par la porte d'entrée :

$$Z_g = w_{xg} \times x + w_{hg} \times h_{t-1} + b_g$$

$$g = \tanh(Z_g)$$

$$Z_j = w_{xi} \times x + w_{hi} \times h_{t-1} + b_i$$

$$i = \text{sigmoid}(Z_i)$$

$$\text{Input_gate_out} = g \times i$$

Passant à travers la porte d'oubli

$$Z_f = w_{xf} \times x + w_{hf} \times h_{t-1} + b_f$$

$$f = \text{sigmoid}(Z_f)$$

$$\text{Forget_gate_out} = f$$

En passant par la porte de sortie :

$$Z_o = w_{xo} \times x + w_{ho} \times h_{t-1} + b_o$$

$$o = \text{sigmoid}(Z_o)$$

$$\text{Out_gate_out} = o$$

- Étape 3 : Calculer la Sortie h_t et l'État Actuel de la Cellule c_t .

$$c_t = (c_{t-1} \times \text{forget_gate_out}) + \text{input_gate_out}$$

$$h_t = \text{out_gate_out} \times \tanh(c_t)$$

Rétro-Propagation Vers l'Arrière (Backward Propagation)

Si nous utilisons MSE (erreur quadratique moyenne) pour l'erreur, alors :

$E = (y - h(x))$, tel que : "y" est la valeur d'origine et $h(x)$ est la valeur prédictive, tel que le gradient transmis par la cellule $E_{\text{delta}} = dE/dh_t$, et à partir de calculer le gradient par rapport à la porte de sortie, ensuite le gradient par rapport à c_t , ensuite le gradient par rapport à la porte d'entrée, ensuite le gradient par rapport à la porte d'oubli, ensuite le gradient par rapport à c_{t-1} ils ont prouvé ces formules montré ci dessous [27] :

Le Gradient par Rapport aux Poids de la Porte de Sortie :

$$dE/dw_{xo} = dE/do \times (do/dw_{xo}) = E_{\text{delta}} \times \tanh(c_t) \times \text{sigmoid}(z_o) \times (1-\text{sigmoid}(z_o)) \times x_t$$

$$dE/dw_{ho} = dE/do \times (do/dw_{ho}) = E_{\text{delta}} \times \tanh(c_t) \times \text{sigmoid}(z_o) \times (1-\text{sigmoid}(z_o)) \times h_{t-1}$$

$$dE/db_o = dE/do \times (do/db_o) = E_{\text{delta}} \times \tanh(c_t) \times \text{sigmoid}(z_o) \times (1-\text{sigmoid}(z_o))$$

Le Gradient Par Rapport aux Poids de la Porte d'Oubli :

$$dE/dw_{xf} = dE/df \times (df/dw_{xf}) = E_{\text{delta}} \times o \times (1-\tanh^2(c_t)) \times c_{t-1} \times \text{sigmoid}(z_f) \times (1-$$

$$\text{sigmoid}(z_f) \times x_t$$

$$\frac{dE}{dw_{hf}} = \frac{dE}{df} \times (\frac{df}{dw_{hf}}) = E_{\text{delta}} \times o \times (1 - \tanh^2(c_t)) \times c_{t-1} \times \text{sigmoid}(z_f) \times (1 - \text{sigmoid}(z_f)) \times h_{t-1}$$

$$\frac{dE}{db_o} = \frac{dE}{df} \times (\frac{df}{db_o}) = E_{\text{delta}} \times o \times (1 - \tanh^2(c_t)) \times c_{t-1} \times \text{sigmoid}(z_f) \times (1 - \text{sigmoid}(z_f))$$

Le Gradient par Rapport aux Poids de la Porte d'Entrée :

$$\frac{dE}{dw_{xi}} = \frac{dE}{di} \times (\frac{di}{dw_{xi}}) = E_{\text{delta}} \times o \times (1 - \tanh^2(c_t)) \times g \times \text{sigmoid}(z_i) \times (1 - \text{sigmoid}(z_i)) \times x_t$$

$$\frac{dE}{dw_{hi}} = \frac{dE}{di} \times (\frac{di}{dw_{hi}}) = E_{\text{delta}} \times o \times (1 - \tanh^2(c_t)) \times g \times \text{sigmoid}(z_i) \times (1 - \text{sigmoid}(z_i)) \times h_{t-1}$$

$$\frac{dE}{db_i} = \frac{dE}{di} \times (\frac{di}{db_i}) = E_{\text{delta}} \times o \times (1 - \tanh^2(c_t)) \times g \times \text{sigmoid}(z_i) \times (1 - \text{sigmoid}(z_i))$$

$$\frac{dE}{dw_{xg}} = \frac{dE}{dg} \times (\frac{dg}{dw_{xg}}) = E_{\text{delta}} \times o \times (1 - \tanh^2(c_t)) \times i \times (1 - \tanh^2(z_g)) \times x_t$$

$$\frac{dE}{dw_{hg}} = \frac{dE}{dg} \times (\frac{dg}{dw_{hg}}) = E_{\text{delta}} \times o \times (1 - \tanh^2(c_t)) \times i \times (1 - \tanh^2(z_g)) \times h_{t-1}$$

$$\frac{dE}{db_g} = \frac{dE}{dg} \times (\frac{dg}{db_g}) = E_{\text{delta}} \times o \times (1 - \tanh^2(c_t)) \times i \times (1 - \tanh^2(z_g)).$$

Ces calculs sont répétés, Jusqu'à trouver les meilleurs paramètres, tel qu'on utilise ces dérivés pour mettre à jour les poids basant sur une fonction de mise à jour (Descente de Gradient Stochastique (**SGD**)) [46] :

$$w^{new} = w^{old} - \alpha \times dw^{old} \quad (4.6)$$

Tel que : α représente le taux d'apprentissage (appelé en anglais learning rate).

Le Taux d'Apprentissage α (Learning Rate)

Le taux d'apprentissage indique la vitesse à laquelle les coefficients évoluent, autrement dit il nous aide à contrôler la rapidité avec laquelle les poids sont mises à jour, sa valeur peut être fixe ou variable, si elle est grande, nous en apprendrons plus, mais nous n'atteindrons peut-être pas la valeur de la plus petite fonction (C-à-dire : on ne peut pas minimiser la fonction de coût "L"). Plus précisément, le taux d'apprentissage est un hyperparamètre configurable utilisé dans l'entraînement des réseaux de neurones qui a une petite valeur positive, souvent comprise entre 0,0 et 1,0.

L'une des méthodes d'optimisation d'entraînement, qui fournissent leurs propres fonctions "loss", les plus populaires s'appelle "Adam", qui a un taux d'apprentissage qui s'adapte au fil du temps [47].

4.2.5 Modes d'Utilisations de LSTM

Bien que les réseaux neuronaux Feed-Forward fassent correspondre une entrée à une sortie, les **RNN** peuvent correspondre à : un à plusieurs, plusieurs à plusieurs et plusieurs à un, il existe

principalement quatre modes pour faire fonctionner un réseau de neurones récurrent de type **LSTM**, chacun de ces modes a son cas d'utilisation [59].

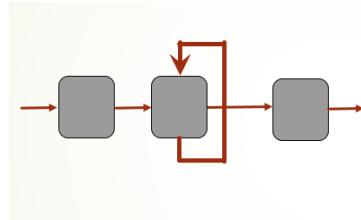


FIGURE 4.12 – Un à un (One to One) [Basé sur [59]]

Un à Un (One to One)

Les problèmes de séquence "Un à un" sont des problèmes de séquence où les données d'entrée ont un pas de temps, et les données de sortie ont un pas de temps [59].

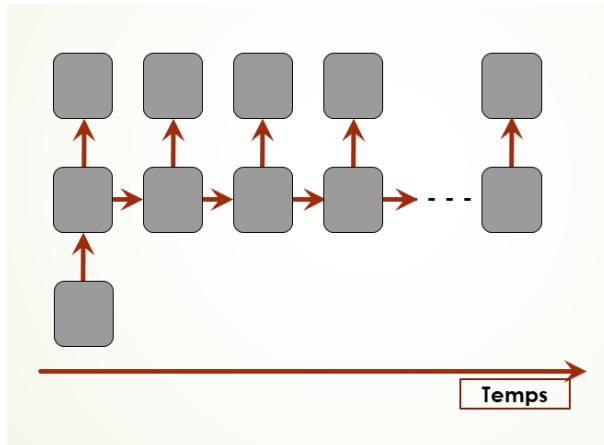


FIGURE 4.13 – Un à Plusieurs (One to Many) [Basé sur [59]]

Un à Plusieurs (One to Many)

Comme montre la figure 4.13, les problèmes de séquence "Un à plusieurs" sont des problèmes de séquence où les données d'entrée ont un pas de temps, et la sortie contient un vecteur de valeurs multiples ou des pas de temps multiples. Ainsi, nous avons une seule entrée et une séquence de sorties [59].

Plusieurs à Un (Many-to-One)

Dans les problèmes de "Plusieurs à un", comme montre la figure 4.14, nous avons une séquence de données en entrée, et nous devons prédire une sortie unique. L'analyse des sentiments ou la classification de textes est un tel cas d'utilisation.

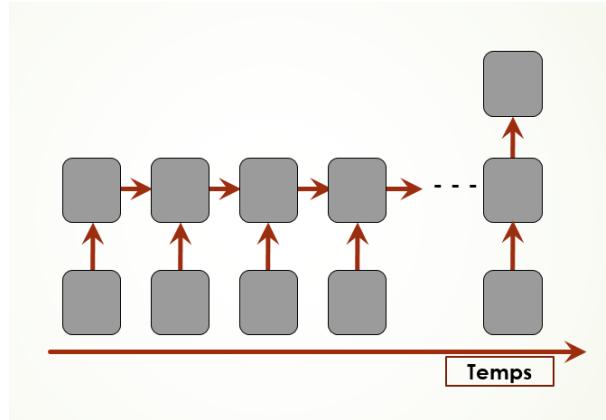


FIGURE 4.14 – Plusieurs à Un (Many-to-One) [Basé sur [59]]

Plusieurs à Plusieurs (Many-to-Many)

L'apprentissage de séquences Plusieurs à plusieurs peut être utilisé pour la traduction automatique, où la séquence d'entrée est dans une certaine langue et la séquence de sortie est dans une autre langue [59].

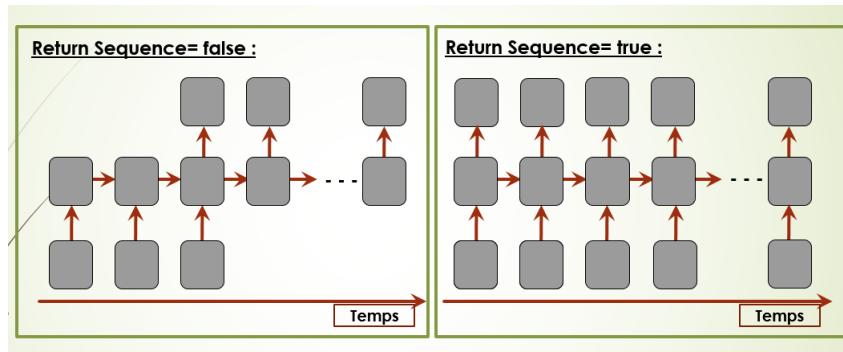


FIGURE 4.15 – Plusieurs à Plusieurs (Many-to-Many) [Basé sur [59]]

Comme expliqué dans les sections Précédentes, les réseaux de neurones récurrents, ou **RNN**, sont spécifiquement conçus pour travailler, apprendre et prédire les données de séquence. où la sortie du réseau d'un pas de temps est fournie comme entrée dans le pas de temps suivant. En outre, les **LSTMs** offrent un certain nombre d'avantages en matière de prévision de séries chronologiques à plusieurs étapes [55] :

- Les **LSTMs** prennent directement en charge plusieurs séquences d'entrée parallèles pour les entrées multivariées, contrairement à d'autres modèles où les entrées multivariées sont présentées dans une structure plate.
- Comme d'autres réseaux de neurones, les **LSTMs** sont capables de mapper les données d'entrée directement sur un vecteur de sortie qui peut représenter plusieurs pas de temps de sortie.
- Des architectures spécialisées ont été développées qui sont spécifiquement conçues pour effectuer des prédictions de séquences en plusieurs étapes, généralement appelées prédiction séquence à séquence (**Seq_To_Seq**) ou plusieurs à plusieurs (**Many To Many**). Un exemple d'architecture de **RNN** conçu pour les problèmes (**Seq_To_Seq**) est l'encodeur-décodeur **LSTM**.

4.2.6 Types des Modèles **LSTM**

4.2.6.1 Modèles **LSTM** Uni-Variés avec une Seule Étape (One Step)

LSTM peut être utilisé pour modéliser des problèmes de prédiction de séries chronologiques univariées. Il existe multiple variations du modèle **LSTM** pour la prévision de séries chronologiques univariées. Parmi ces modèles, on a :

LSTM Simple (Vanilla LSTM)

Le modèle **LSTM** simple est composé d'une seule couche **LSTM** cachée suivie d'une couche de sortie standard à anticipation [52].

LSTM Empilé (Stacked LSTM)

LSTM empilé se compose de plusieurs couches **LSTM** cachées où chaque couche contient plusieurs cellules de mémoire. L'empilement de couches cachées **LSTM** rend le modèle plus profond [52].



FIGURE 4.16 – Archiecture de Mémoire à Long Terme Empilée [Basé sur [49]]

LSTM Bidirectionnels (BLSTM)

Dans la plupart des problèmes d'analyse de séquence, il est intéressant d'avoir une mémoire du passé pour prendre de bonnes décisions à l'instant t.

Mais dans certains problèmes de reconnaissance, il est parfois intéressant de regarder les observations futures, lorsque celles-ci sont disponibles [52] [45].

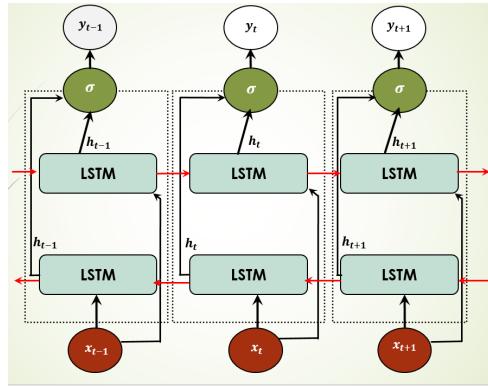


FIGURE 4.17 – Architecture de LSTM Bidirectionnels [Basé sur [12]]

Exemple Illustratif, [45]

Dans une phrase pour deviner un mot caché, On peut analyser le contexte passé, mais aussi

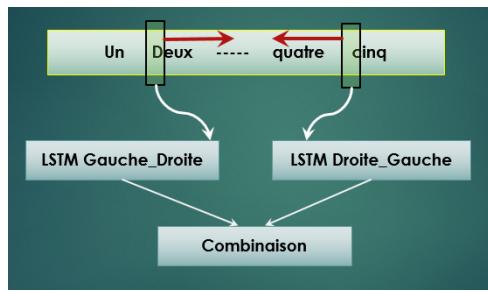


FIGURE 4.18 – LSTM Bidirectionnels [Basé sur [13]]

le contexte futur. Pour ces problèmes, il est possible d'utiliser un réseau appelé BLSTM, qui consiste à dédoubler une couche LSTM, l'une étant apprise pour parcourir le signal de gauche à droite, et l'autre de droite à gauche.

4.2.6.2 Modèles LSTM Multivariés avec Plusieurs Étapes (Multi-Steps)

Il existe différents modèles LSTM qui peuvent être utilisés pour les problèmes de prédiction des séries chronologiques qui nécessitent une prédiction de plusieurs pas de temps dans le futur, parmi eux, nous citons le modèle Encodeur-Décodeur.

Le Modèle Encodeur-Décodeur

Comme son nom l'indique, le modèle est composé de deux sous-modèles : l'encodeur et le décodeur. L'encodeur est un modèle responsable de la lecture et de l'interprétation de la séquence d'entrée. La sortie de l'encodeur est un vecteur de longueur fixe qui représente l'interprétation de la séquence par le modèle. L'encodeur est traditionnellement un modèle Vanilla LSTM, bien que d'autres modèles d'encodeurs puissent être utilisés, tels que les modèles empilés, bidirectionnels

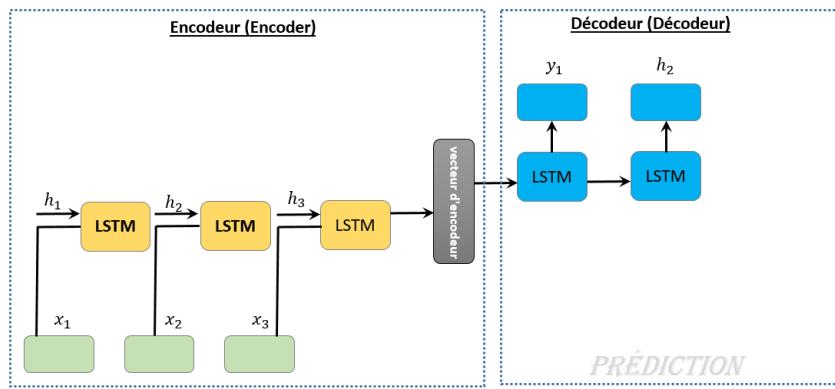


FIGURE 4.19 – Architecture du Modèle Encodeur-Décodeur [basé sur [50]]

et CNN.

Comme déjà indiqué, l'encodeur **LSTM** joue le même rôle de lecture de la séquence d'entrée et de génération des vecteurs : (h_k, c_k) . Cependant, le décodeur doit prédire la séquence de sortie entière compte tenu des vecteurs : (h_k, c_k) [50] [52].

4.2.7 Préparation des Données pour les Réseaux **LSTM**

Pour bien comprendre comment préparer les données pour les réseau **LSTM**, on va présenter les critères sur lesquels reposent les résultats des modèle **LSTM** en expliquant la relation entre le modèle et les données d'entraînement par rapport à ces critères.

La Fenêtre de Prédiction (TimeSteps)

La préparation des données pour le réseau **LSTM** comprend des pas de temps. Certains problèmes de séquence peuvent avoir un nombre varié de pas de temps par échantillon. Par exemple, étant donné le temps courant (t) nous voulons prédire une valeur au temps suivant dans la séquence ($t + 1$), nous pouvons utiliser le temps courant (t), ainsi que les deux temps précédents ($t - 1$ et $t - 2$) comme variables d'entrée.

Comme montre la figure 4.20, il y a une compatibilité entre le nombre de neurones dans le réseau **LSTM** et la taille de la fenêtre de prédiction, dans les expériences d'utilisation d'un seul neurone la capacité d'apprentissage du réseau est limitée, généralement l'augmenter du nombre de neurones dans le **LSTM** (C-à-dire : l'augmentation des pas de temps) se traduit par une augmentation des performances Plus précisément le "TimeSteps" signifie combien de valeurs existent dans une séquence [30].

La Taille du Vecteur "h" (Nombres d'Unités en **LSTM**)

Comme montre la figure 4.21, chaque cellule **LSTM** cachée est composée de plusieurs unités

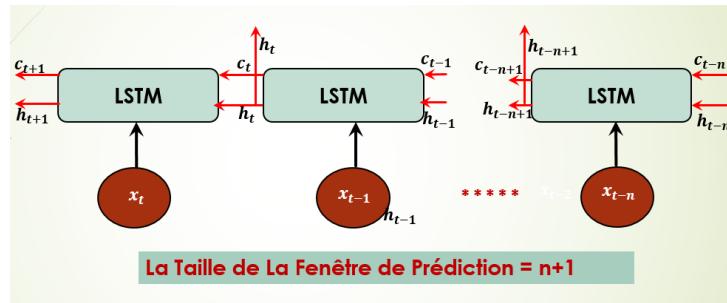


FIGURE 4.20 – La Fenêtre de Prédiction (TimeSteps) [Basé sur [75]]

cachées, Le nombre d'unités est en fait la dimension de l'état caché (ou la dimension de la sortie " h_t "). En général, Plus le nombre d'unités est grand (plus grande dimension d'états cachés), plus le réseau devient capable à se souvenir de motifs plus complexes. Le nombre d'unités définit la dimension des états cachés (ou sorties) et le nombre de paramètres dans la couche **LSTM** [78].

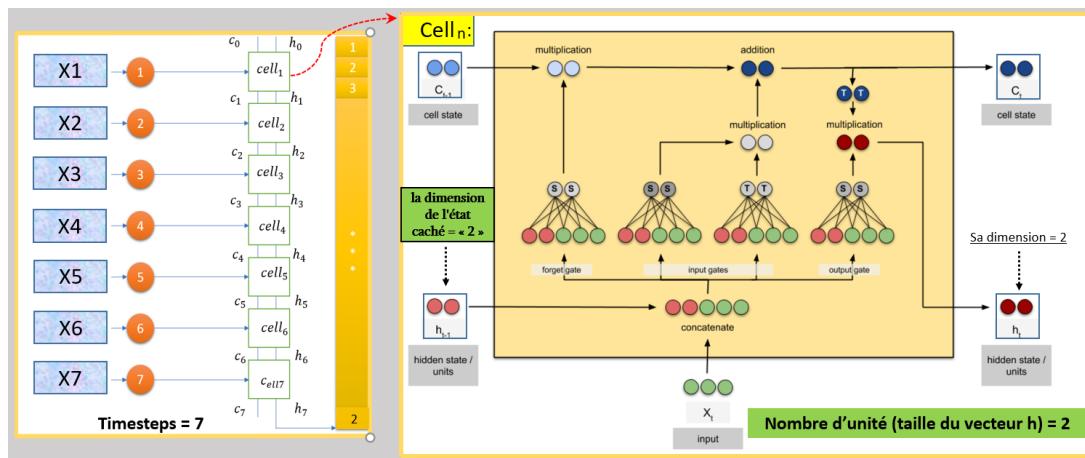


FIGURE 4.21 – Nombres d'unités en LSTM [Basé sur [44]]

Taille du Lot (Le Batch Size)

Comme montre la figure 4.22, la taille du lot est un hyperparamètre qui définit le nombre d'échantillons à traiter avant de mettre à jour les paramètres du modèle interne en utilisant rétro-propagation. On peut considérer un Taille du Lot comme une boucle *for* itérant sur un ou plusieurs échantillons et effectuant des prédictions, à la fin du lot, les prédictions sont comparées aux variables de sortie attendues et une erreur est calculée, à partir de cette erreur, l'algorithme de mise à jour est utilisé pour améliorer le modèle, par exemple "descendre le long du gradient d'erreur".

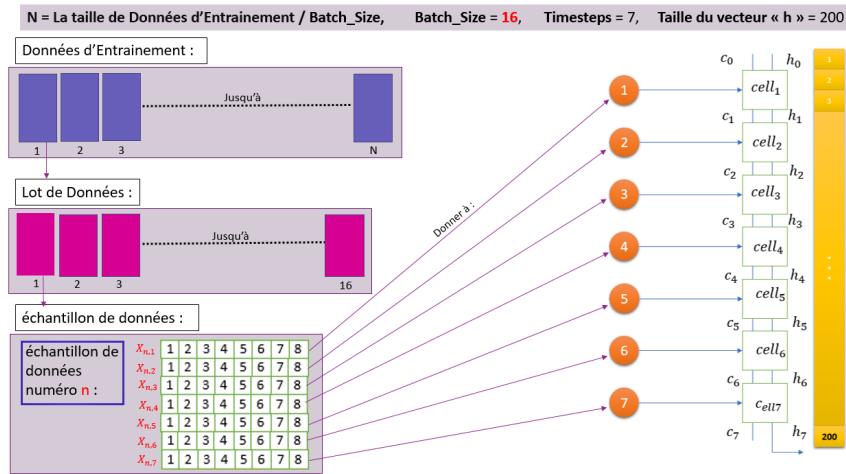


FIGURE 4.22 – Le Batch Size (Taille du Lot) [Basé sur [11]]

Nombre d'Époque (Epoch)

Comme montre la figure 4.23, une époque signifie que chaque échantillon de l'ensemble de données d'apprentissage a eu l'occasion de mettre à jour les paramètres du modèle interne. Une époque est composée d'un ou plusieurs lots. Par exemple, comme montre la figure 4.23 une époque qui a un lot est appelée algorithme d'apprentissage par descente de gradient par lots.

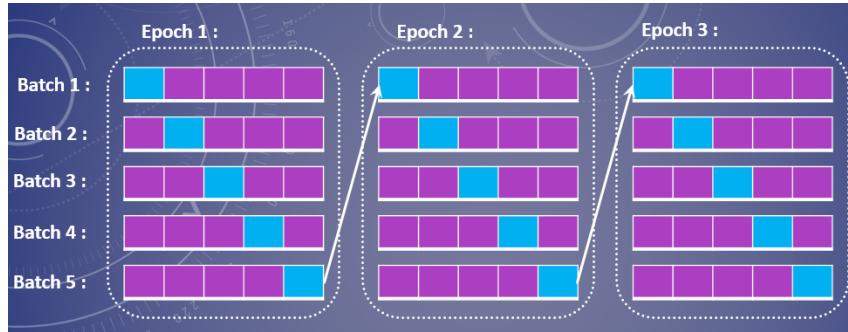


FIGURE 4.23 – Nombre d'Époques (Epoch) [Basé sur [28]]

4.3 Conclusion

Dans ce chapitre, nous avons présenté une vue globale sur l'un des systèmes et technologies permettant de concevoir et réaliser notre approche MEAP, qui est la méthode de prédiction LSTM.

Dans le chapitre suivant, on va présenter les travaux connexes afin de montrer la valeur ajoutée dans notre approche MEAP.

Chapitre 5

Travaux connexes

Contents

5.1	Introduction	44
5.2	Travaux Connexes	44
5.2.1	Proactive Autoscaling for Cloud-Native Applications using Machine Learning	44
5.2.2	Agnostic Approach for Microservices Autoscaling in Cloud Applications	45
5.2.3	An Improved Kubernetes Scheduling Algorithm For Deep Learning Platform	45
5.2.4	An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud	46
5.2.5	Adaptive scaling of Kubernetes Pods LIBRA	47
5.2.6	Proactive Autoscaling for Edge Computing Systems with Kubernetes	47
5.2.7	A lightweight autoscaling mechanism for fog computing in industrial applications	47
5.2.8	Quantifying Cloud Elasticity with Container-Based Autoscaling	48
5.2.9	A study on performance measures for auto-scaling CPU-intensive containerized applications	48
5.2.10	Building an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows	48
5.2.11	Autoscaling Pods on an On-Premise Kubernetes Infrastructure QoS-Aware	48
5.3	Analyse Comparative	48
5.4	Conclusion	49

5.1 Introduction

ans ce chapitre on va présenter une étude bibliographique sur plusieurs travaux connexes dans la littérature dans la Section 5.2, et nous terminons par une analyse comparative, dans la Section 5.3, afin de montrer la valeur ajoutée de notre approche MEAP.

5.2 Travaux Connexes

5.2.1 Proactive Autoscaling for Cloud-Native Applications using Machine Learning

Dans l'un des premiers articles sur le thème de mise à l'échelle automatique et prédictive, Nicolas Marie-Magdelaine et Toufik Ahmed [56] ont proposé la mise à l'échelle proactive pour les applications cloud natives utilisant Machine Learning qui est basé sur L'approche architecturale des microservices qui consiste à développer l'application comme une collection de petits services. Chacun est indépendant et implémente atomique fonctionnalités. Les demandes de services aux entreprises sont traitées par plusieurs microservices interagissant les uns avec les autres. Comme ils sont composants indépendants, des microservices peuvent être déployés, mis à niveau, mis à l'échelle et redémarré indépendamment, permettant ainsi des versions rapides et fréquentes sur les applications en direct avec peu ou pas d'impact sur les utilisateurs finaux.

Conteneurisé les microservices offrent plus d'efficacité et de rapidité que le virtuel ceux des machines. Les conteneurs peuvent être instanciés avec la vitesse et facilité de tout processus de SE. Plusieurs conteneurs peuvent s'exécuter sur le même SE et bénéficient de l'isolation entre eux. Cette capacité à mutualiser les ressources et réduire les frais généraux fait des conteneurs le facteur de forme idéal pour microservices. Les conteneurs offrent également une flexibilité avec l'API de gestion qui permet une automatisation complète de leur cycle de vie. Des fonctionnalités telles que des orchestrateurs effectuant des mises à jour continues, la mise à l'échelle automatique de base et bien d'autres peuvent utiliser l'API.

Les méthodes proposées dans ces travaux présentent des limites qu'ils s'agit principalement : les applications basées sur des conteneurs et des microservices sont plus volatiles, plus difficiles à surveiller et présentent les problèmes d'allocation des ressources principalement en terme d'isolation. Par rapport aux applications de microservices basées sur des MV, les applications cloud natives sont basées sur des conteneurs partageant une infrastructure sous-jacente, gérée de manière dynamique et basée sur le millage de services, découverte de services, messagerie, etc., qui

rendent ce système distribué complexe à observer et orchestrer.

5.2.2 Agnostic Approach for Microservices Autoscaling in Cloud Applications

Abeer Abdel Khaleq, Ilkyeon Ra [38] adaptent la méthode de mise à l'échelle automatique Kubernetes basée sur les microservices. On teste l'effet d'autres métriques sur le temps de réponse des microservices non consommateurs de CPU.

L'évolutivité du conteneur par défaut de K8s est basée sur l'utilisation du processeur des Pods. Cela fonctionnera bien pour une application consommatrice de CPU. Cependant, les applications ont des différences besoins en ressources et différentes demandes de qualité de service. Ce travail présente l'approche agnostique pour adapter le K8s autoscaler basé sur les exigences spécifiques du microservice, et tester cette approche sur des applications non gourmandes en CPU et microservices utilisant différentes métriques telles que la mémoire et nombre de messages non distribués dans la file d'attente. Ces tests montrent une amélioration du temps de réponse du microservice par rapport à la mise à l'échelle automatique du processeur par défaut.

5.2.3 An Improved Kubernetes Scheduling Algorithm For Deep Learning Platform

Dans cet article [34] SHI HUAXIN, GU XIAOFENG, KUANG PING, HUANG HONGYU proposent une solution multi-tenant pour améliorer l'Algorithme de planification K8s orienté modèle. Basé sur la stratégie de planification HiveDScheduler qui modélise les utilisateurs comme clusters virtuels, on mesure la situation de charge du cluster périodiquement, le Pod sur le noeud est programmé pour effectuer l'équilibrage de charge des clusters. Cette méthode améliore la stabilité et la disponibilité de la plateforme, et fonctionnent bien dans des scénarios de développement collaboratif multi-équipes.

HiveDScheduler partant du principe que les utilisateurs de plusieurs équipes utilisent le même pool de ressources, HiveDScheduler modélise différents utilisateurs dans différents clusters virtuels, et gère les ressources de calcul pour chaque cluster virtuel. Les utilisateurs au sein du cluster ont le plus haute priorité et peuvent utiliser les ressources allouées à volonté. Lorsqu'un cluster virtuel a un degré élevé d'inactivité, les utilisateurs en dehors du cluster peuvent également utiliser les ressources informatiques du cluster pour travailler, mais leur priorité est moindre. Cette conception peut répondre aux besoins des utilisateurs de plusieurs équipes de partager le même pool de ressources tout en améliorant l'utilisation des ressources.

L’expérience montre que cette méthode peut répondre aux besoins des utilisateurs des différentes équipes, et détecter la charge du cluster pour assurer l’équilibrage de charge périodiquement.

5.2.4 An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud

Mulugeta Ayalew Tamiru, Johan Tordsson, Erik Elmroth, Guillaume Pierre [73] présentent le Cluster Autoscaler (**CA**) qui ajuste dynamiquement le nombre et la taille des **MV** sur lesquelles les conteneurs fonctionnent. Comme les autres composants **K8s**, **CA** est hautement configurable. Dans sa configuration par défaut, **CA** ajoute ou supprime les nœuds identiques.

Cependant, **K8s** a récemment introduit une fonctionnalité de Provisionnement Automatique des Nœuds (**NAP**) qui ajoute automatiquement des nœuds à partir de plusieurs pools de nœuds. Contrairement à la plupart des autoscalers à la pointe de la technologie, Cluster Autoscaler-Provisionnement Automatique des Nœuds (**CA-NAP**) permet le provisionnement dynamique de nœuds de tailles différentes. Ceci est particulièrement utile lorsque certain Pods ont des plus grande demande de ressources que le reste des Pods dans la charge de travail. **CA-NAP** peut alors provisionner des nœuds spécifiquement correspondant à la demande de ces Pods. De plus, il a le potentiel pour des économies significatives dans les clouds publics en sélectionnant les **MV** de la bonne taille pour correspondre à la charge de travail.

Les principaux résultats de leurs expériences approfondissent sur les performances de mise à l’échelle automatique de **CA** et **CA-NAP** :

- **CA-NAP** surpassé **CA**, car il provisionne nœuds de tailles différentes pour répondre à la demande du meilleure charge de travail ;
- **CA-NAP** ne propose pas économie de coûts significative par rapport à **CA** ;
- La performance de **CA-NAP** est influencée principalement par la composition de la charge de travail, plus performante pour charges de travail composées de plusieurs durées courtes et longues Pods avec diverses demandes de ressources ;
- **CA** et **CA-NAP** affichent un surapprovisionnement pire mais meilleure précision de sous-provisionnement en temps partagé ;
- **CA** et **CA-NAP** pourraient offrir des performances encore meilleures si les autres paramètres de configuration tels que l’intervalle de mise à l’échelle automatique, le temps de mise à l’échelle automatique sont réglé correctement.

5.2.5 Adaptive scaling of Kubernetes Pods LIBRA

Les algorithmes de mise à l'échelle automatique deviennent de plus en plus mis en avant dans le domaine du cloud computing. Dans cet article, l'auteur a proposé un autoscaler adaptatif Libra, qui détecte automatiquement l'ensemble de ressources optimaux pour un seul Pod, gère ensuite le processus de la mise à l'échelle automatique horizontale. De plus, si la charge ou l'environnement virtualisé sous-jacent change, Libra adapte la définition de ressource pour le Pod et ajuste le processus de mise à l'échelle horizontale en conséquence.

Afin de fournir un mécanisme de mise à l'échelle automatisé dans K8s Balla, Csaba Simon, Markosz Maliosz [10] proposent Libra, un autoscaler pour les applications s'exécutant sur K8s. L'idée principale derrière Libra est que les algorithmes de mise à l'échelle automatique traditionnels prennent le niveau de mise à l'échelle de l'utilisateur et déclenchent un événement de mise à l'échelle au cas où la consommation de ressource dépasse ces limites. Cependant, l'attribution de la bonne mise à l'échelle des niveaux des services n'est pas triviale pour les opérateurs humains. Habituellement, ces niveaux de mise à l'échelle sont définis après une surveillance de longue durée de services.

Le problème avec cette approche est l'apparition de nouvelles évolutions. Le comportement de ces nouveaux services est inconnu par les opérateurs, un niveau de mise à l'échelle raisonnable ne peut donc pas être attribué.

5.2.6 Proactive Autoscaling for Edge Computing Systems with Kubernetes

Dans cet article [36], les auteurs ont proposé un autoscaler proactif de Pod multi-métriques et personnalisable. L'autoscaler est capable de collecter plusieurs métriques, de prévoir les charges de travail et de mettre à l'échelle les applications cibles à l'avance. De plus, il permet aux utilisateurs de personnaliser leurs propres politiques de mise à l'échelle et modèles de prédiction pour mieux s'adapter à leurs applications.

5.2.7 A lightweight autoscaling mechanism for fog computing in industrial applications

Les auteurs dans [76] ont combiné hyperviseur et virtualisation conteneurisée pour la plate-forme Fog afin de déployer, gérer et mettre à l'échelle automatiquement les applications industrielles. Les auteurs ont utilisé la logique floue pour construire la méthode de mise à l'échelle automatique afin de réduire les coûts opérationnels et d'améliorer les performances des applica-

tions. Cependant, la méthode proposée est réactive et basée sur des règles, ce qui ne permet pas d'assurer des garanties de temps de réponse.

5.2.8 Quantifying Cloud Elasticity with Container-Based Autoscaling

Les auteurs dans [74] ont discuté des différentes mesures de performance pour la mise à l'échelle automatique des applications basées sur des conteneurs sous la charge de travail intensive en CPU.

5.2.9 A study on performance measures for auto-scaling CPU-intensive containerized applications

Les auteurs dans [16] ont proposé un cadre pour mettre à l'échelle automatiquement les applications conteneurisées en surveillant l'utilisation des ressources des conteneurs et gérant la charge de travail fluctuante.

5.2.10 Building an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows

Les auteurs dans [39] ont proposé une méthode de mise à l'échelle automatique réactive pour mettre à l'échelle les applications bioinformatique et biomédicales hébergées sur le cloud. La méthode proposée utilise la mise à l'échelle automatique horizontale pour améliorer les performances de l'application.

5.2.11 Autoscaling Pods on an On-Premise Kubernetes Infrastructure QoS-Aware

Les auteurs dans [71] ont présenté une architecture sur site basée sur des conteneurs Kubernetes et Docker visant à améliorer la **QoS** concernant l'utilisation des ressources et les objectifs de niveau de service **SLO**. La principale contribution de cette proposition est ses capacités de mise à l'échelle dynamique pour ajuster les ressources système à la charge de travail actuelle tout en améliorant la qualité de service.

5.3 Analyse Comparative

La Table 5.1 présente une analyse comparative entre notre approche proposée **MEAP** et les différentes approches analysées précédemment, en terme de : 1) prédition (Préd), 2) une méthode de prédition efficace telle que **LSTM**, 3) la modularité (Mod) où le système de la mise à l'échelle

automatique est modulaire et extensible de façon que le module de prédiction est séparé, 4) le modèle mathématique (Modèle), 5) l'utilisation d'un cluster réel (Cluster), et finalement, 6) la proposition d'un algorithme.

TABLE 5.1 – Analyse Comparative entre notre Contribution **MEAP** et les différents Travaux Connexes Analysés précédemment

Référence	Préd	LSTM	Mod	Modèle	Cluster	Algorithme
[MARIE-MAGDELAINE et AHMED 2020]	✓	✓	✗	✗	✗	✓
[KHALEQ et RA 2019]	✗	✗	✗	✗	✗	✓
[HUAXIN et al. 2020]	✗	✗	✗	✓	✓	✗
[TAMIRU et al. 2020]	✗	✗	✗	✗	✓	✗
[BALLA, SIMON et MALIOSZ 2020]	✗	✗	✗	✗	✓	✗
[JU, SINGH et TOOR 2021]	✓	✗	✗	✓	✗	✓
[TSENG et al. 2018]	✗	✗	✗	✗	✗	✗
[TANG et al. 2017]	✗	✗	✗	✗	✗	✗
[CASALICCHIO 2019]	✓	✗	✗	✓	✗	✓
[KRIEGER et al. 2017]	✗	✗	✗	✗	✗	✗
[RUÍZ et al. 2022]	✗	✗	✗	✗	✗	✓
Notre Contribution MEAP	✓	✓	✓	✓	✓	✓

Different de ces travaux qui présentent plusieurs limites, l'approche **MEAP** considère tous ces aspects cités au dessus, comme montre la dernière ligne de la Table 5.1, ce qui montre la valeur ajoutée de notre contribution.

5.4 Conclusion

Dans ce chapitre, nous avons élaboré une étude bibliographique sur plusieurs approches de mise à l'échelle automatique proposées dans la littérature. Ensuite, nous avons présenté une analyse comparative entre ces approches et notre approche **MEAP** selon plusieurs critères fondamentaux. Cette analyse comparative montre la valeur ajoutée de notre approche.

Dans le chapitre suivant, on va présenter l'architecture globale de l'approche **MEAP** ainsi que le modèle prédictif **LSTM**, et nous terminons par un modèle mathématique ainsi qu'une heuristique.

L'Approche de Mise à l'Échelle Automatique et Prédictive (MEAP) - Conception

Contents

6.1	Introduction	51
6.2	Architecture Globale de MEAP	51
6.3	Le Modèle Prédictif Choisi	53
6.4	Formulation Mathématique du Problème Correspondant à MEAP	54
6.5	L'Algorithme de Mise à l'Échelle Automatique et Prédictive MEAP	56
6.6	Conclusion	57

6.1 Introduction

Dans ce chapitre, on va montrer la conception de l'approche MEAP, dans lequel on va commencer par une vue globale sur les différents composants de son architecture, comme montré dans la Section 6.2, la conception du modèle prédictif selon le type Encodeur-Décodeur de la méthode de prédiction LSTM, comme montré dans la Section 6.3. Ensuite, nous présentons un modèle mathématique généralisant l'approche MEAP, comme montré dans la Section 6.4, et finalement, pour résoudre ce modèle mathématique, nous terminons par la proposition de l'Algorithm 1, comme montré dans la Section 6.5.

6.2 Architecture Globale de MEAP

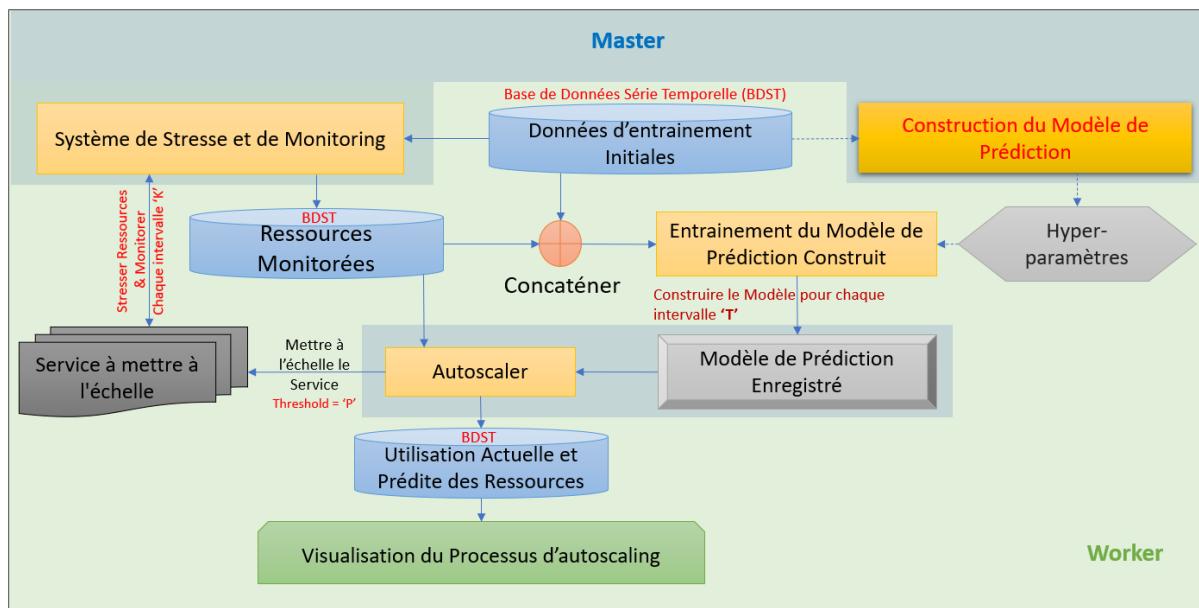


FIGURE 6.1 – Architecture Globale de MEAP

La figure 6.1, représente l'architecture globale de l'approche MEAP, elle se compose de plusieurs modules, qui sont répartis entre le noeud master et les noeuds workers d'un certain orchestrateur.

1. **Données d'Entraînement Initiales** : représente un modèle de données que le module "Système de Stresse et de Monitoring" doit utiliser pour générer un modèle d'utilisation de ressources réel.
2. **Système de Stresse et de Monitoring** : il permet, dans un premier temps, de stresser des ressources d'un microservice (Service à Mettre à l'Échelle), tel que (CPU, Mémoire), ce qui permet de forcer leurs utilisation de suivre un certain modèle de données. Ensuite, il

va monitorer l'utilisation de ces ressources et l'enregistre dans une base de données temps réel déployée dans le système d'orchestration (Ressources Monitorées).

3. Construction du Modèle de prédiction : ce module permet, via les données d'entraînement initiales, de trouver les meilleurs paramètres du modèle de prédiction, tels que : les hyperparameters d'un réseau de neurones récurrent comme **LSTM** (le nombre de neurones, le nombre de couches, les types de couches, le nombre d'époque d'entraînement, etc.). Ces paramètres seront utilisés par le module "Entraînement du Modèle de Prédiction Construit".
4. Entraînement du Modèle de Prédiction Construit : il permet, via les paramètres fournis par le module "Construction du Modèle de prédiction", d'entraîner d'une manière continue et périodique (la période correspond au : TimeSteps × intervalle de monitoring) le modèle prédictif en utilisant comme entrée "Données d'Entraînement Initiales" concaténées avec les données monitorées (Ressources Monitorées). Le modèle entraîné sera sauvegardé dans un module séparé (Modèle de Prédiction Enregistré).
5. Modèle de Prédiction Enregistré : correspond au modèle de prédiction entraîné et enregistré dans un fichier séparé, pour qu'il soit utilisé comme entrée pour le module "Autoscaler".
6. Ressources Monitorées : correspond à l'utilisation des ressources sauvegardées dans une base de données temps réel.
7. Autoscaler : il permet de mettre à l'échelle un certain microservice (Service à Mettre à l'Échelle), si l'utilisation prédictive des ressources sera supérieure à un certain seuil. Ce module utilise comme entrée les ressources monitorées (Ressources Monitorées) pour prédire la future utilisation des ressources en utilisant le modèle entraîné (Modèle de Prédiction Enregistré).
8. Service à Mettre à l'Échelle : correspond au microservice sur lequel on doit appliquer le stress, le monitoring, la mise à l'échelle automatique.
9. Utilisation Actuelle et Prédite des Ressources : Ce module correspond aux données à visualiser.
10. Visualisation de Processus de mise à l'échelle automatique : Il permet de montrer l'évolution des processus de monitoring, prédiction et mise à l'échelle.

6.3 Le Modèle Prédicatif Choisi

Par référence à l'étude qu'on a élaboré dans les chapitres "Vue Globale sur les Différentes Technologies Utilisées", "État de l'Art", notre choix s'est tombé sur le type "Encodeur-Décodeur" de la méthode de prédiction **LSTM**.

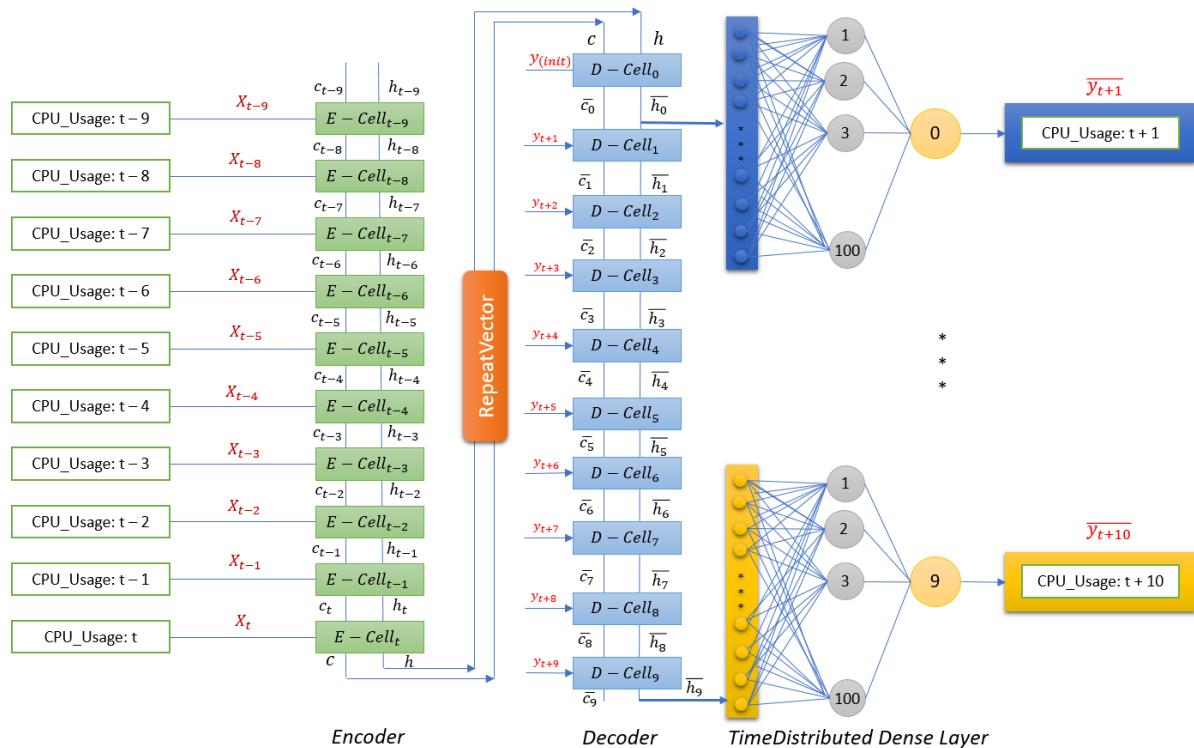


FIGURE 6.2 – Architecture du Modèle de Prédiction Utilisé (**LSTM** Encodeur-Décodeur)

La figure : 6.2 représente l'adaptation du type "Encodeur-Décodeur" à notre cas d'utilisation, qui est la prédiction de l'utilisation de CPU. Le choix du type "Encodeur-Décodeur" se justifie d'un coté, par son bonne précision de prédiction, et d'un autre coté par son compatibilité avec la conception de notre cas d'utilisation. En effet, l'objectif est de prédire une séquence d'utilisations de **CPU** à partir d'une séquence d'utilisation de **CPU** dans le passé, le type "Encodeur-Décodeur" permet de réaliser cette conception.

Le modèle **LSTM** "Encodeur-Décodeur" montré dans la figure : 6.2, prend comme entrée une séquence d'utilisation de **CPU** ($CPU_Usage : t-n, CPU_Usage : t-(n-1), \dots, CPU_Usage : t$). Chaque valeur de cette séquence avec le vecteur h ainsi que le vecteur c , représentent les entrées d'une cellule **LSTM**, chaque cellule **LSTM** est connectée récursivement avec une autre cellule précédente, l'ensemble de ces cellules représentent la première couche du modèle **LSTM**, ce qu'on appelle : "Encodeur". En fait, cette couche donne comme sortie deux vecteurs : c, h , qui représente une sorte d'encodage de la séquence d'entrée, d'où vient le nom "Encodeur".

Il est à noter que, une couche **LSTM** prend la séquence d'utilisation de **CPU** avec la dimension (Taille_Batch, TimeSteps, Features), et comme la sortie de la première couche, qui est h, c , est de dimension (taille du vecteur h , TimeSteps), il est nécessaire de répéter le vecteur h afin d'assurer les trois dimensions pour la deuxième couche **LSTM**.

Ensuite, une deuxième couche **LSTM** qui prend comme entrée les vecteurs " h, c ", prédire les futures utilisations de **CPU**, d'où vient le nom "Décodeur". Chaque cellule **LSTM** donne comme sortie les vecteurs h, c . Ensuite, un nouveau réseau de neurones va prendre comme entrée ce dernier vecteur h , pour interpréter une seule valeur du **CPU** prédictive. En effet, chaque cellule **LSTM** est complètement connectée (dense) à un réseau de neurones séparé, son nombre égale à "TimeSteps".

Il est à noter que, la mise à jour des poids (W, b) des cellules de la couche Encodeur, est connue sous le nom **Back-Propogation Over Time**, où chaque cellule a ses propres poids qui sont différents de ceux des autres cellules, la mise à jour se fait dans la première cellule : $E-Cell_t$ et se propage vers la plus loin (la dernière cellule : $E-Cell_{t-9}$), d'où le nom "*Over Time*". D'un autre coté, la mise à jour des poids (W, b) des cellules de la couche Décodeur, se fait en suivant la méthode **Teacher Forcing**, qui permet d'utiliser dans l'entraînement les valeurs réelles de **CPU** (y_t) pour faire la prédiction, ensuite, les poids (W, b) sont mis à jour au fur et à mesure qu'on fait la prédiction, contrairement à la couche Encodeur, où on doit attendre la dernière prédiction dans la séquence.

6.4 Formulation Mathématique du Problème Correspondant à MEAP

Un cluster Kubernetes est représenté par : $C = (M, W, MS)$, dont :

- M : représente la liste des noeuds Masters, $M = (m_i, i \in [1, |M|])$
- W : représente la liste des noeuds Workers, $W = (w_i, i \in [1, |M|])$
- MS : représente la liste des MicroServices déployés dans le cluster "C", $MS = (ms_i, i \in [1, |MS|])$

Un MicroService " ms_i " est représenté par :

$ms_i = (S_{ms_i}, D_{ms_i}, V_{ms_i}, P_{ms_i}, R_{ms_i})$, dont :

- S_{ms_i} : représente l'ensemble des services utilisés par le MicroService " ms_i " pour exposer à l'extérieur.
- D_{ms_i} : représente l'ensemble de déploiements pour le MicroService " ms_i ".
- V_{ms_i} : représente l'ensemble de volumes utilisés par les Pods des MicroServices " ms_i ".

- P_{ms_i} : représente l'ensemble de Pods pour le MicroService " ms_i ".
- R_{ms_i} : représente l'ensemble de replicaset utilisés pour assurer que le nombre de replicas P_{ms_i} reste le même.

Afin de définir l'objectif de notre modèle, on va commencer par définir les variables suivantes :

- $CPU_{ms_i}^{P_j}$: représente l'utilisation de CPU par le Pod " P_j " qui fait partie du MicroService " ms_i ".
- $\overline{CPU}_{ms_i}^{P_j}$: représente l'utilisation prédictive de CPU par le Pod " P_j " qui fait partie du MicroService " ms_i ".

- Le nombre de replicas d'un MicroService durant l'intervalle " T_s " (TimeSteps) est représenté par :

$$R_{ms_i}^{T_s} = \left| \frac{\sum_{j=1}^{|P_j|} CPU_{ms_i}^{P_j}}{\gamma} \right| \quad (6.1)$$

dont : " γ " représente le seuil maximal d'utilisation de **CPU** (Threshold), en dépassant ce seuil, le Pod commence à être surchargé.

- L_{ms_i} : représente le nombre de fois que l'utilisation du CPU ($CPU_{ms_i}^{P_j}$) arrive ou dépasse le seuil γ .

- θ : représente le seuil de confiance de prédiction, il correspond au nombre de fois qu'on prédit l'utilisation du **CPU** dépasse le seuil γ dans l'intervalle T_s . La mise à l'échelle se déclenche après qu'on prédit le dépassement du seuil γ , θ fois.

- $Scale_D^{T_s}$: permet de détecter si le nombre de prédiction de dépassement du seuil γ , atteint le seuil θ . Il prend deux valeurs (1 ou 0),

$$Scale_D^{T_s} = \sum (\overline{CPU}_{ms_i}^{P_j} > \gamma) \geq \theta = \{1 \text{ sinon } 0\} \quad (6.2)$$

Objectif :

$$\min(\psi \cdot \sum_{j=1}^{|P_j|} .CPU_{ms_i}^{P_j} + \beta \cdot L_{ms_i}). \quad (6.3)$$

Contraintes :

1. Limitations de nombre de replicas : $|R_{ms_i}^{T_s}| < \omega$.
2. Chevauchement : $P_{ms_i} \cap P_{ms_j} = \emptyset, \forall (ms_i, ms_j) \in MS^2$.
3. Équilibrage de charge : $|CPU_{ms_i}^{P_j} - CPU_{ms_i}^{P_k}| < \lambda, \forall (P_j, P_k) \in P_{ms_i}^2$.

L'objectif de ce modèle mathématique est de minimiser la consommation de **CPU** et le nombre de fois que l'utilisation de **CPU** atteint le seuil γ , comme montré dans l'équation 6.4, en respectant différentes contraintes, notamment : 1) Limitations de nombre de replicas : l'objectif de cette contrainte est de limiter le nombre de replicas par microservice, afin de ne pas affecter les

performances des autres microservices, dont le paramètre ω représente le nombre maximal du replicas par microservice, 2) Chevauchement : assure que l'ensemble des Pods n'appartiennent pas à d'autre microservices, 3) Équilibrage de charge : assure l'équilibrage de la charge après la mise à l'échelle automatique d'un microservice, dont le paramètre λ représente une petite valeur représentant la différence de la charge entre l'utilisation du **CPU** de différents Pods d'un même microservice.

Il est à noter que les deux paramètres ψ , β représentent les poids des différentes parties de la fonction objective, en effet, pour donner plus de poids à l'utilisation du **CPU**, ψ doit être plus grand que β . Sinon, pour limiter le nombre de surchargement d'un Pod on peut donner plus d'importance au paramètre β .

6.5 L'Algorithme de Mise à l'Échelle Automatique et Prédictive MEAP

Algorithm 1: Algorithm **MEAP**

```

1: procedure (Timestep :  $T_s$ , Interval :  $I$ )
2:   for  $P_j \in ms_i$  do
3:      $S \leftarrow 1$ 
4:      $R \leftarrow 1$ 
5:     for each  $(T_s \times I)$  do
6:       if  $\overline{CPU}_{ms_i}^{P_j} > \gamma$  then
7:          $S \leftarrow S + 1$ 
8:       end if
9:       if  $S > \theta$  then
10:         $Scale_D^{T_s}$ 
11:         $R \leftarrow \text{Min}(R + 1, \omega)$ 
12:      end if
13:      if  $S < \theta$  and  $\overline{CPU}_{ms_i}^{P_j} > \gamma$  then
14:         $L_{ms_i} \leftarrow L_{ms_i} + 1$ 
15:      end if
16:    end for
17:  end for
18: end procedure

```

Pour résoudre le modèle mathématique proposé dans la Section 6.4, nous proposons l'Algorithm 1. Cet algorithme fonctionne comme suit : il prend comme entrée le TimeSteps T_s et l'intervalle I , il va vérifier pour tous les Pods P_j faisant partie d'un microservice ms_i , comme montré dans la ligne (5), si l'utilisation prédictive du **CPU** du Pods : P_j ($\overline{CPU}_{ms_i}^{P_j}$) dépasse le threshold γ , il va incrémenter la variable S , comme montré dans la ligne (6), si le nombre de fois qu'on prédit le dépassement du threshold γ dépasse le seuil de confiance θ , il va déclencher

une mise à l'échelle, comme montré dans les lignes : (9 :11), ainsi comme on doit respecter la contrainte du limitation du nombre de replicas d'un même microservice, la ligne (11) permet de limiter ce nombre à ω .

Finalement, le dernier bloc "if" permet de compter le nombre de fois que l'utilisation **CPU** actuelle $CPU_{ms_i}^{P_j}$ dépasse le threshold γ .

6.6 Conclusion

Dans ce chapitre, nous avons présenté l'architecture globale du système **MEAP**, ensuite, nous avons présenté la conception de notre modèle prédictif, permettant de prédire l'utilisation de **CPU** selon le type Encodeur-Décodeur de la méthode de prédiction **LSTM**. Après, nous avons montré le modèle mathématique correspondant à l'approche proposée **MEAP**, pour résoudre ce modèle mathématique, nous avons proposé l'algorithme **MEAP**.

Dans le chapitre suivant, on va présenter la plateforme de **MEAP**, ensuite, on va discuter les différents résultats.

Tests et Résultats

Contents

7.1	Introduction	59
7.2	Plateforme de Mise à l'Échelle Automatique et Prédictive	59
7.2.1	Scénario et Procédure de Mise à l'Échelle Automatique	59
7.2.2	Technologies et Outils Utilisés Pour le Déploiement de MEAP	72
7.2.3	Procédure de Déploiement du Système de Mise à l'Échelle Automatique et Prédictive	73
7.3	Résultats	79
7.3.1	La méthode Root Mean Square Error (RMSE)	79
7.3.2	Évaluation du Modèle LSTM	79
7.3.3	Évaluation de l'approche de Mise à l'Échelle Automatique et Prédictive MEAP	86
7.4	Conclusion	88

7.1 Introduction

Dans ce chapitre, nous évaluons l'efficacité de notre approche **MEAP**. Nous commençons par la présentation de la plateforme de mise à l'échelle automatique et prédictive dans la Section 7.2, dans laquelle nous présentons le Scénario de simulation dans la Section 7.2.1. Ensuite, la liste des technologies utilisées dans la Section 7.2.2, et la procédure de déploiement de **MEAP** dans la Section 7.2.3. Finalement, nous présentons l'évaluation de l'approche **MEAP** dans la Section 7.3.

7.2 Plateforme de Mise à l'Échelle Automatique et Prédictive

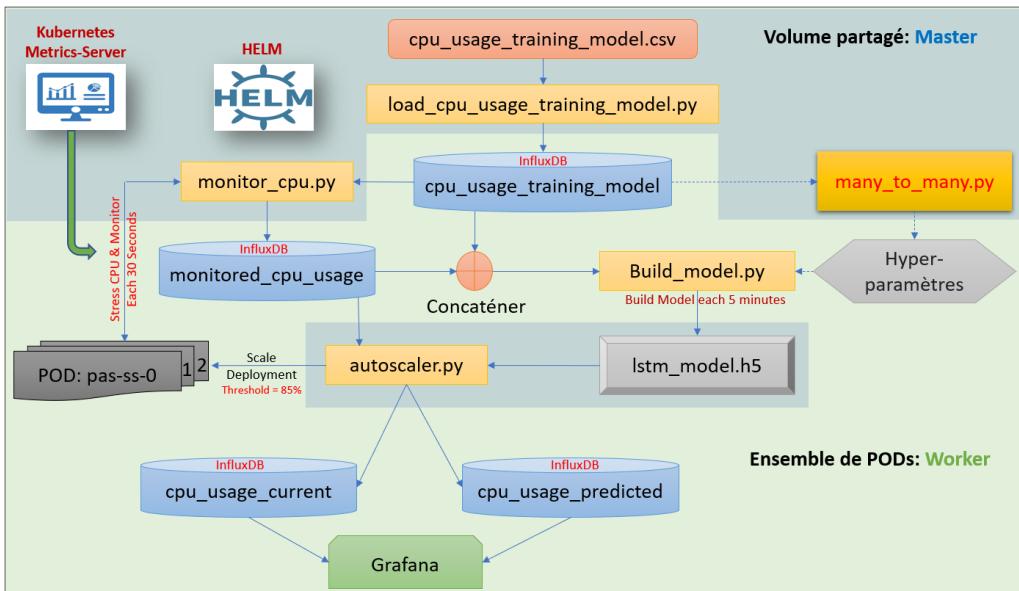


FIGURE 7.1 – Plateforme de Mise à l'Échelle Automatique et Prédictive

7.2.1 Scénario et Procédure de Mise à l'Échelle Automatique

Dans cette section, on va présenter la plateforme montrée dans la figure 7.1, dont, on va expliquer le rôle de chaque composant ou module, ainsi l'interconnexion entre ces modules.

Afin de montrer l'efficacité et la précision de prédiction de notre modèle **LSTM** Encodeur-Décodeur, nous avons simulé manuellement un modèle de données représentant l'utilisation de **CPU**. Ensuite, nous avons forcé le **CPU** du Pod "pas-ss-0" à être chargé en suivant exactement ce modèle, et comme nous avons conçu l'entraînement contenu du modèle **LSTM**, le modèle simulé s'accumule continuellement avec des consommations réelles de **CPU**.

Initialement, le modèle simulé contient des évolutions qui dépassent un certain seuil de consommation de **CPU** que nous avons fixé dans notre plateforme à 85%. L'idée c'est de mettre

à l'échelle "pas-ss-0" proactivement avant que l'utilisation de **CPU** atteint ce seuil. Également, nous avons fixé le nombre maximum de replicas, après mise à l'échelle automatique, à 5, et ce pour n'est pas affecter les ressources des autres microservices.

La figure 7.1 montre une description détaillée du scénario montré ci-dessus, et comme notre approche **MEAP** est modulaire, elle se compose de plusieurs modules. Dans ce qui suit, on va détailler ces modules :

cpu_usage_training_model.csv :

Il correspond à un fichier Comma Separated Values (**CSV**) contenant le modèle simulé de l'utilisation de **CPU** qu'on doit utiliser pour stresser le **CPU** du Pod : "pas-ss-0".

load_cpu_usage_training_model.py :

C'est un script Python permettant de charger les données depuis le fichier "CSV" : "**cpu_usage_training_model.csv**" vers le measurement d'influxdb : "**cpu_usage_training_model**". Dans ce qui suit, on va expliquer les fonctions composants ce script :

La fonction **connect** permet de connecter avec le serveur influxdb :

```
def connect(host='localhost', port=8086):
    user = 'admin'
    password = 'admin'
    dbname = 'pas'
    dbuser = 'admin'
    dbuser_password = 'admin'
    client = InfluxDBClient(host, port, user, password, dbname)
    return client
```

La fonction **insert_point** permet d'insérer un seul enregistrement dans un measurement :

```
def insert_point(timestamp, cpu_usage, measurement):
    json_body = [
        {
            "measurement": measurement,
            "tags": {
                "host": "pas-ss-0"
            },
            "time": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "fields": {
                "cpu_usage": cpu_usage
            }
        }
    ]
    result = client.write_points(json_body)
    return result
```

On fait appel à la fonction **connect** pour se connecter au serveur influxdb, ici, l'adresse IP 10.111.219.45 corresponde à l'adresse du service influxdb du Kubernetes :

```
print("connecting to influxd: ")
client = connect("10.111.219.45", 8086)
```

Les instructions suivantes servent à lire le modèle depuis le fichier CSV :

`cpu_usage_training_model.csv` et enregistrer les données dans la variable dataset :

```
dataset = read_csv('cpu_usage_training_model.csv', header=0,
                   infer_datetime_format=True, parse_dates=['datetime'],
                   index_col=['datetime'])
data = dataset.values
data = array(data)
```

Une fois les données récupérées dans la variable "dataset", ainsi que la connexion avec influxdb est initialisée, les instructions suivantes servent à insérer ces données dans le measurement `cpu_usage_training_model` :

```
timestamp = datetime.now() - timedelta(hours=11, minutes=40, seconds=0)
for index in range(len(data)):
    insert_point(timestamp, data[index], "cpu_usage_training_model")
    timestamp = timestamp + timedelta(hours=0, minutes=0, seconds=30)
```

cpu_usage_training_model

Il correspond à un measurement dans influxdb, qui sert à sauvegarder les données du modèle simulé d'utilisation de **CPU**.

monitor_cpu.py

Il correspond à un script Python, qui permet de stresser le **CPU** du Pod "pas-ss-0" par le modèle enregistré dans `cpu_usage_training_model`, et ce en utilisant l'outil "stress-ng" déjà installé au niveau du Pod "pas-ss-0". Ensuite, on collecte la consommation de **CPU** en utilisant le module "Metrics-Server" du Kubernetes. Dans ce qui suit, on détaille les fonctions construisant ce script :

La fonction `get_training_data` permet de récupérer le modèle d'entraînement depuis le measurement : `cpu_usage_training_model` :

```
def get_training_data():
    query = 'select cpu_usage from cpu_usage_training_model order by time asc'
    result = client.query(query)
    data = []
    for point in result.get_points():
        data.append(point['cpu_usage'])
    return data
```

Afin de communiquer entre le module `autoscaler.py`, qui change l'état de la mise à l'échelle périodiquement, et le module `monitor_cpu.py`, nous avons créé un measurement "scalestate" qui sauvegarde l'état de la mise à l'échelle. Ensuite, on peut récupérer cette état ("True","False") via la fonction `get_state` :

```
def get_state():
    query = 'select scale from scalestate'
    result = client.query(query)
    for point in result.get_points():
        state = point['scale']
    return eval(str(state))
```

La fonction `update_state` permet de modifier l'état de la mise à l'échelle :

```
def update_state(scale=False):
    client.query('drop measurement scalestate')
    timestamp = datetime.now()
    json_body = [
        {
            "measurement": "scalestate",
            "tags": {
                "host": "pas-ss-0"
            },
            "time": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "fields": {
                "scale": scale
            }
        }
    ]
    result = client.write_points(json_body)
    return result
```

Comme on fait le `stress`, ensuite le `monitoring` chaque 30s, à chaque valeur monitorée, on appelle la fonction `insert_cpu_usage` pour insérer un seul enregistrement dans le measurement `monitored_cpu_usage` :

```
def insert_cpu_usage(cpu_usage, timestamp):
    json_body = [
        {
            "measurement": "monitored_cpu_usage",
            "tags": {
                "host": "pas-ss-0"
            },
            "time": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "fields": {
                "cpu_usage": cpu_usage
            }
        }
    ]
    result = client.write_points(json_body)
```

La fonction `stress_cpu_monitor` permet d'exercer un stresse sur le Pod "pas-ss-0" d'un pourcentage : "percentage" et pendant une durée : "duration". Ensuite, elle fait le monitoring, en utilisant le module "Metrics-Server" du Kubernetes, via la commande : "kubectl top Pod | grep pas-ss-0" :

```

def stress_cpu_monitor(percentage = 50, duration = 10):
    try:
        proc_stress = subprocess.Popen("kubectl exec -namespace=default
                                       pas-ss-0 - bash -c 'stress-ng -c 1 -l "+str(percentage)+" -t
                                       "+str(duration)+"s'", shell=True, stdin=subprocess.PIPE,
                                       stdout=subprocess.PIPE, preexec_fn=os.setsid)
        cpu_usage_list = []
        for i in range(10):
            proc_monitor = subprocess.Popen("kubectl top Pod | grep
                                             pas-ss-0", shell=True, stdout=subprocess.PIPE)
            cpu_usage_line = proc_monitor.communicate()[0]
            cpu_usage_line = str(cpu_usage_line)
            index = 10
            cpu_usage = ""
            while cpu_usage_line[index] == 'm':
                cpu_usage += cpu_usage_line[index]
                index += 1
            cpu_usage_list.append(float(cpu_usage))
            time.sleep(3)
            proc_monitor.kill()
        proc_stress.kill()
        cpu_usage_core = int(1000 * (percentage / 100))
        return adjust_measurement(cpu_usage_list, cpu_usage_core)
    except:
        return 0

```

La fonction `init_all` permet d'initialiser le nombre de replicas, l'état du scaling à "False", et supprimer le measurement : `monitored_cpu_usage` :

```

def init_all():
    proc1 = subprocess.Popen("kubectl scale sts pas-ss -replicas=1", shell=True,
                           stdout=subprocess.PIPE)
    update_state(False)
    clear_cpu_usage()
    return proc1

```

Les instructions suivantes permettent : 1) Initialiser le cluster, 2) Récupération du modèle de simulation de l'utilisation de **CPU** dans la variable "data" :

```

client = connect("10.111.219.45", 8086)
print("Please wait while initializing the cluster")
proc1 = init_all()
data = get_training_data()
time.sleep(30)
proc1.kill()
print("Cluster initialized successfully")

```

Les instructions suivantes font appel à la fonction `stress_cpu_monitor` qui va stresser et monter le modèle `cpu_usage_training_model`, après, ils vont sauvegarder la consommation de **CPU** dans le measurement `monitored_cpu_usage`. De plus, ils vont mettre à jour l'état de la mise à l'échelle :

```

i = 0
while True:
    state = get_state()
    if state:
        i = random.randint(0, int(len(data)/1000) - 1) * 100
        update_state(False)
    print("la valeur de i est: ", i)
    print("CPU will be loaded by: ", data[i], " %")
    int(data[i]), 30)
    cpu_usage = str(int(((stress_cpu_monitor(int(data[i]),
30))*100)/1000))
    print("cpu_usage: ", cpu_usage, " %")
    "monitored_cpu_usage" dans "influxdb"
    timestamp = datetime.now()
    insert_cpu_usage(cpu_usage, timestamp)
    i = (i + 1) % len(data)

```

monitored_cpu_usage

Il correspond à un measurement dans influxdb, dans lequel on sauvegarde au fur et à mesure la consommation monitorée du **CPU**, et qu'elle sera utilisé, par la suite, par le module autoscaler.

Build_model.py

Il correspond à un script Python, il prend comme entrée la concaténation de la consommation de **CPU** actuellement monitorée, depuis le measurement "monitored_cpu_usage", ainsi que le modèle de simulation de la consommation de **CPU** initiale. Ensuite, il va répartir ces données entre données d'entraînement (train) et données de test (test), une fois le modèle est entraîné, il sera sauvegardé dans un fichier séparé "lstm_model.h5", également, il sera évalué en utilisant la méthode **RMSE**. Il est à noter que l'entraînement, la sauvegarde du modèle entraîné et l'évaluation de ce modèle se font périodiquement, d'où la notion de continuité de ce module. Dans ce qui suit, on va détailler les fonctions constituant ce module. Rappelant que le TimeSteps correspond au nombre de pas qu'on fait dans le passé pour prédire une séquence de même taille (TimeSteps), l'instruction suivante sert à initialiser ce paramètre à "10" :

La fonction `get_training_data` permet de récupérer le modèle d'entraînement, elle consiste à concaténer les données sauvegardé dans le measurement "cpu_usage_training_model" et le measurement "monitored_cpu_usage" :

```

def get_training_data():
    data = []
    query_cpu_usage_training_model = 'select cpu_usage from          cpu_usage_training_model order
by time asc'
    result_cpu_usage_training_model =
    client.query(query_cpu_usage_training_model)
    for point in result_cpu_usage_training_model.get_points():
        data.append(int(point['cpu_usage']))
    query_monitored_cpu_usage = 'select cpu_usage from monitored_cpu_usage          order by time
asc'
    result_monitored_cpu_usage = client.query(query_monitored_cpu_usage)

```

```

for point in result_monitored_cpu_usage.get_points():
    data.append(int(point['cpu_usage']))
data = array(data)
data = data.reshape(len(data), 1)
return data

```

La fonction `split_dataset` permet de répartir la consommation de **CPU** entre données d'entraînement et données de test :

```

def split_dataset(data):
    up_to_ts = len(data) % timestep
    if up_to_ts == 0:
        train, test = data[:up_to_ts], data[-(100 + up_to_ts):-up_to_ts]
        train = array(split(train, len(train)/timestep))
        test = array(split(test, len(test)/timestep))
    else:
        train, test = data, data[-100:]
        train = array(split(train, len(train)/timestep))
        test = array(split(test, len(test)/timestep))
    return train, test

```

La fonction `evaluate_forecasts` permet d'évaluer le modèle de prédiction **LSTM** entraîné. Elle consiste à calculer le **RMSE** globale dans "score" ainsi que le **RMSE** par période dans "scores" dont scores est un vecteur de taille "TimeSteps". Le **RMSE** par période consiste à estimer la précision de prédiction entre chaque valeur prédictée et réelle de l'utilisation de **CPU**, tandis que le **RMSE** globale consiste à estimer la précision de prédiction entre toutes les valeurs prédictées et réelles à la fois :

```

def evaluate_forecasts(actual, predicted):
    scores = list()
    for i in range(actual.shape[1]):
        mse = mean_squared_error(actual[:, i], predicted[:, i])
        rmse = sqrt(mse)
        scores.append(rmse)
    s = 0
    for row in range(actual.shape[0]):
        for col in range(actual.shape[1]):
            s += (actual[row, col] - predicted[row, col])**2
    score = sqrt(s / (actual.shape[0] * actual.shape[1]))
    return score, scores

```

La fonction `summarize_scores` permet d'afficher le **RMSE** globale et pour chaque pas de temps :

```

def summarize_scores(name, score, scores):
    s_scores = ', '.join(['%.1f' % s for s in scores])
    print('%s: %.3f %s' % (name, score, s_scores))

```

Comme la taille des données d'entraînement n'est pas suffisante, la fonction `split_dataset` permet encore de minimiser cette taille car on regroupe ces données d'entraînement sous des

vecteurs de taille "TimeSteps", ce qui mène à avoir un modèle d'entraînement pauvre en terme de taille du modèle. De ce fait, on construit des vecteurs par un décalage d'une position à chaque fois, par exemple : au lieu d'avoir $[1,2,3,4],[5,6,7,8]$ on construit les données d'entraînement de la façon suivante : $[1,2,3,4],[2,3,4,5],[3,4,5,6], \dots$ etc. Ensuite, on doit répartir ces vecteurs entre deux groupes "X" correspondant au données réelles et "Y" correspondant au données qu'on doit prédire de la façon suivante : $[1,2,3,4] \in "X"$ et $[2,3,4,5] \in "Y"$, ainsi de suite. Cette répartition corresponde à construire un modèle d'entraînement supervisé : $f(X) = Y$:

```
def to_supervised(train, n_input, n_out=timestep):
    l'instruction suivante permet de flatten
    data = train.reshape((train.shape[0]*train.shape[1], train.shape[2]))
    X, y = list(), list()
    in_start = 0
    10 dans X
    position: in_start += 1
    for _ in range(len(data)):
        in_end = in_start + n_input
        out_end = in_end + n_out
        if out_end <= len(data):
            x_input = data[in_start:in_end, 0]
            x_input = x_input.reshape((len(x_input), 1))
            X.append(x_input)
            y.append(data[in_end:out_end, 0])
        in_start += 1
    return array(X), array(y)
```

La fonction `build_model` permet de construire le modèle **LSTM** en précisant plusieurs paramètres, elle retourne le modèle après l'entraînement dans : "return model", aussi elle permet de sauvegarder le modèle entraîné dans un fichier : "lstm_model.h5" :

```
def build_model(train, n_input):
    train_x, train_y = to_supervised(train, n_input)
    verbose, epochs, batch_size = 0, 2, 16
    n_timesteps, n_features, n_outputs = train_x.shape[1], train_x.shape[2],
    train_y.shape[1]
    train_y = train_y.reshape((train_y.shape[0], train_y.shape[1], 1))
    model = Sequential()
    model.add(LSTM(200, activation='relu', input_shape=(n_timesteps,
    n_features)))
    model.add(RepeatVector(n_outputs))
    model.add(LSTM(200, activation='relu', return_sequences=True))
    model.add(TimeDistributed(Dense(100, activation='relu')))

    model.add(TimeDistributed(Dense(1)))
    model.compile(loss='mse', optimizer='adam')
    model.fit(train_x, train_y, epochs=epochs, batch_size=batch_size,
    verbose=verbose)
    model.save('lstm_model.h5')
    return model
```

La fonction `forecast` permet de faire la prédiction en utilisant le modèle déjà entraîné. Le paramètre "history" représente tous les vecteurs dans "train", c'est une liste de vec-

teurs, pour faire la prédiction, on prend juste les 10 dernières valeurs dans `input_x`: `input_x = data[-n_input:, 0]`, ensuite, on utilise le modèle entraîné pour faire la prédiction dans le vecteur "yhat" :

```
def forecast(model, history, n_input):
    data = array(history)
    data = data.reshape((data.shape[0]*data.shape[1], data.shape[2]))
    input_x = data[-n_input:, 0]
    input_x = input_x.reshape((1, len(input_x), 1))
    yhat = model.predict(input_x, verbose=0)
    yhat = yhat[0]
    return yhat
```

La fonction `evaluate_model` permet d'évaluer le modèle **LSTM**, en utilisant `test`. En fait, `test` est un vecteur de 10 vecteurs, et chaque sous vecteur contient 10 valeurs. L'idée de cette fonction est d'utiliser les 10 dernières valeurs de `train` c'est à dire, le dernier vecteur dans `train` pour prédire les 10 premières valeurs de `test` c'est à dire le premier vecteur dans `test`. Ensuite, elle va rajouter le vecteur prédit dans `train` via l'instruction : `history.append(test[i,:])`, pour avoir un nouveau vecteur d'entraînement : `history`, ensuite, elle utilise le dernier vecteur dans `history`, c'est à dire, le vecteur prédit précédemment pour prédire le deuxième vecteur dans `test`, jusqu'à arrivée au vecteur "10" :

```
def evaluate_model(model, train, test, n_input):
    history = [x for x in train]
    predictions = list()
    for i in range(len(test)):
        yhat_sequence = forecast(model, history, n_input)
        predictions.append(yhat_sequence)
        history.append(test[i,:])
    predictions = array(predictions)
    score, scores = evaluate_forecasts(test[:, :, 0], predictions)
    return score, scores
```

Les instructions suivantes permettent de : 1) Connecter à influxdb, 2) Récupération des données d'entraînement comme expliqué précédemment, 3) entraînement du modèle **LSTM**, et 4) Évaluation du modèle entraîné :

```
client = connect('192.168.171.85', 8086)
while True:
    data = get_training_data()
    print("La taille du modèle d'entraînement: ", len(data))
    train, test = split_dataset(data)
    print("La taille du modèle d'entraînement: ", len(train))
    n_input = timestep * 1
    model = build_model(train, n_input)
```

```

score, scores = evaluate_model(model, train, test, n_input)
model.summary()
summarize_scores('lstm', score, scores)
print("Le modèle à été entraîné et sauvegardé sur      : 'lstm_model.h5'")
time.sleep(60)

```

many_to_many.py

Il correspond à un script Python, qui consiste, de manière similaire au module "Build_model", à entraîner un modèle prédictif **LSTM** et l'évaluer, la seule différence est que le module "many_to_many" est utilisé juste pour trouver les meilleurs hyperparamètre tel que le nombre d'époques (epoch), taille de lot (batch size), la taille du vecteur "h", pour qu'ils soient utilisé dans le modèle construit "Build_model".

lstm_model.h5

Pour séparer l'entraînement continu du modèle **LSTM** au module autoscaler, nous avons sauvegardé le modèle entraîné dans un fichier séparé, ce dernier sera utilisé par le script "autoscaler.py".

autoscaler.py

Il correspond à un script Python, qui prend comme entrée : 1) La consommation du **CPU** actuellement monitorée, à partir du measurement "*monitored_cpu_usage*", 2) Le modèle **LSTM** entraîné et sauvegardé dans le fichier "lstm_model.h5". À partir d'une séquence de consommation de **CPU** de taille "TimeSteps", récupéré depuis le measurement "*monitored_cpu_usage*", le modèle **LSTM** sauvegardé prédit une séquence de taille "TimeSteps". Si une valeur dans la séquence prédite dépasse le seuil 85%, ce module déclenche la mise à l'échelle prédictive, en communiquant avec le cluster Kubernetes via "Kubectl". Il est à noter que, ce script s'exécute périodiquement. Dans ce qui suit, on va détailler les fonctions de ce script.

La fonction `insert_replicas` permet d'insérer le nombre de replicas du microservice "pas-ss-0" :

```

def insert_replicas(replicas, timestamp):
    json_body = [
        {
            "measurement": "replicas",
            "tags": {
                "host": "pas-ss-0"
            },
            "time": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "fields": {
                "replicas": replicas
            }
        }
    ]
    result = client.write_points(json_body)
    return result

```

La fonction `insert_input_x` permet d'insérer le vecteur `input_x` dans le measurement : "cpu_usage_current", ce vecteur est de taille "TimeSteps". L'objectif de cette fonction est de sauvegarder la consommation de CPU actuelle pour la visualiser sur "Grafana" :

```
def insert_input_x(input_x, timestamp):
    for cpu_usage in input_x:
        json_body = [
            {
                "measurement": "cpu_usage_current",
                "tags": {
                    "host": "pas-ss-0"
                },
                "time": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
                "fields": {
                    "cpu_usage": cpu_usage
                }
            }
        ]
        result = client.write_points(json_body)
        timestamp = timestamp - timedelta(hours=0, minutes=0,
                                           seconds=30)
    return result
```

La fonction `insert_yhat` permet d'insérer le vecteur "yhat" dans le measurement : "cpu_usage_predicted", ce vecteur est de taille "TimeSteps". L'objectif de cette fonction est de sauvegarder la consommation de CPU prédicté pour la visualiser sur "Grafana" :

```
def insert_yhat(yhat, timestamp):
    for cpu_usage in yhat:
        timestamp = timestamp + timedelta(hours=0, minutes=0,
                                           seconds=30)
        json_body = [
            {
                "measurement": "cpu_usage_predicted",
                "tags": {
                    "host": "pas-ss-0"
                },
                "time": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
                "fields": {
                    "cpu_usage": cpu_usage
                }
            }
        ]
        result = client.write_points(json_body)
    return result
```

La Fonction `get_input` permet de récupérer les données actuellement monitorées depuis le measurement "monitored_cpu_usage" :

```
def get_input():
    query = 'select cpu_usage from monitored_cpu_usage order by time asc'
    result = client.query(query)
    data = []
    for point in result.get_points():
        data.append(int(point['cpu_usage']))
    return data[-timestep:]
```

La fonction `forecast` permet de prédire une séquence de taille "TimeSteps" dans le vecteur "yhat", en donnant comme entrée, le vecteur `input_x` représentant une séquence de taille "TimeSteps" dans `cpu_usage` :

```
def forecast(model, inpt_list, n_input):
    data = array(inpt_list)
    data = data.reshape(timestep, 1)
    input_x = data[-n_input:, 0]
    input_x = input_x.reshape((1, len(input_x), 1))
    yhat = model.predict(input_x, verbose=0)
    yhat = yhat[0]
    return input_x, yhat
```

La Fonction `is_scale_out` retourne "True" si une des valeurs de la séquence prédictes dépasse le seuil 85% :

```
def is_scale_out(yhat, threshold):
    autoscale = False
    for i in range(len(yhat)):
        if yhat[i] > threshold:
            autoscale = True
    return autoscale
```

La fonction `scale` permet de mettre à l'échelle le microservice "pas-ss-0" :

```
def scale(replicas=1):
    proc1 = subprocess.Popen("kubectl scale sts pas-ss
                            -replicas="+str(replicas)+"", shell=True, stdout=subprocess.PIPE)
    return proc1
```

La fonction `update_state` permet de modifier l'état de la mise à l'échelle, pour qu'elle soit utilisée par le module "monitor_cpu.py" :

```
def update_state(scale=False):
    client.query('drop measurement scalestate')
    timestamp = datetime.now()
    json_body = [
        {
            "measurement": "scalestate",
            "tags": {
                "host": "pas-ss-0"
            },
            "time": timestamp.strftime("%Y-%m-%d %H:%M:%S"),
            "fields": {
                "scale": scale
            }
        }
    ]
    result = client.write_points(json_body)
    return result
```

La fonction `init_all` permet d'initialiser l'état de la mise à l'échelle ainsi que l'ensemble de measurements : `cpu_usage_current` , `cpu_usage_predicted`. Elle est exécutée à chaque fois on lance le module "autoscaler" :

```
def init_all():
    update_state(False)
    client.query('drop measurement cpu_usage_current')
    client.query('drop measurement cpu_usage_predicted')
    client.query('drop measurement replicas')
    timestamp = datetime.now()
    insert_replicas(1, timestamp)
```

L'instruction suivante permet d'initialiser le module "autoscaler" :

```
init_all()
```

`cpu_threshold` c'est le seuil maximale d'utilisation du **CPU**, à partir de ce seuil, on peut considérer que le conteneur est surchargé :

```
cpu_threshold = 85
```

Initialisation du nombre de replicas à 1 :

```
replicas = 1
```

Les instructions suivantes consistent à : 1) Charger une séquence de taille "TimeSteps" : "X", pour qu'elle sera l'entrée du modèle **LSTM** (**LSTM(X)=Y**), 2) Prédiction de la séquence futur "Y", 3) Si une valeur de "Y" dépasse le seuil "85%" on fait appel à la fonction "scale" pour mettre à l'échelle le microservice "pas-ss-0", 4) Charge les deux measurements "cpu_usage_current" et "cpu_usage_predicted", pour visualiser sur "Grafana" :

```
while True:
    inpt_list = get_input()
    if len(inpt_list) == timestep:
        n_input = timestep * 1
        model = load_model('lstm_model.h5')
        input_x, yhat = forecast(model, inpt_list, n_input)
        print("yhat = LSTM_MODEL[input_x]")
        print("les ", timestep, " dernières valeurs de cpu_usage input_x
        :\n", input_x)
        print("les ", timestep, " valurs de cpu_usage à prédir yhat
        :\n", yhat)
        input_x = input_x.reshape(10, 1)
        input_x = [x[0] for x in input_x]
        input_x = [input_x[len(input_x) - i] for i in range(1,
        len(input_x) + 1)]
        timestamp = datetime.now()
```

```

insert_input_x(input_x, timestamp)
insert_yhat(yhat, timestamp)
replicas = get_replicas()
if is_scale_out(yhat, cpu_threshold):
    replicas = 1 + (replicas % 5)
    update_state(True)
    insert_replicas(replicas, timestamp)
    proc1 = scale(replicas)
    time.sleep(2)
    proc1.kill()
    print("pas-ss has been scaled succesfully !!!")
else:
    insert_replicas(replicas, timestamp)
else:
    print("La taille du vecteur: input_x is < ", timestep, " !!",
          yhat = LSTM_MODEL[input_x]")
time.sleep(300)

```

cpu_usage_current

Il correspond à un measurement dans influxdb, il est utilisé pour visualiser la consommation actuelle du **CPU** sur "Grafana".

cpu_usage_predicted

Il correspond à un measurement dans influxdb, il est utilisé pour visualiser la consommation prédictive du **CPU** sur "Grafana".

Pod : pas-ss-0

Il correspond au microservice sur lequel on applique le mécanisme de la mise à l'échelle automatique.

7.2.2 Technologies et Outils Utilisés Pour le Déploiement de **MEAP**

Pour implémenter et déployer notre solution proposée **MEAP**, nous avons installé un cluster Kubernetes version "v1.24.3", ce cluster est composé d'un noeud master et un autre noeud worker, les deux installés sur deux machines virtuelle, en utilisant l'hyperviseur "VirtualBox", et "Ubuntu" version "20.04". Le cluster Kubernetes installé utilise comme "Container Runtime" : Docker. Ainsi, les Pods dans les différents noeuds du cluster, communiquent entre eux via le plugin réseau "calico".

Comme montre la figure 7.1, nous avons utilisé l'**API** Kubernetes "kubectl" pour déployer : 1) Le microservice sur lequel on va appliquer la mise à l'échelle automatique "pas-ss-0", 2) L'outil de visualisation et monitoring "Grafana", 3) Le microservice "build_model" afin t'entraîner le modèle **LSTM** d'une manière continue, au fur et à mesure qu'on reçoit de nouvelles données monitorées. D'un autre coté, nous avons utilisé l'outil "Helm", pour installer la base de données

série temporelle "influxdb", en effet, l'outil "Helm" permet l'automatisation de l'installation des applications sur Kubernetes. Également, nous avons utilisé des volumes de stockage persistants pour : "influxdb, Grafana et build_model" , afin de garder leur configurations. Pour collecter des statistiques sur l'utilisation du **CPU** et Mémoire, nous avons activé "Metrics-Server" au sein de Kubernetes.

Pour le développement des scripts de stresse et monitoring, de mise à l'échelle automatique et d'entraînement du modèle **LSTM**, nous avons utilisé le langage de programmation "Python". De plus, pour construire le modèle de prédiction **LSTM**, nous avons utilisé les bibliothèques : "Keras", "TensorFlow" et "Numpy". Il est à noter que, nous avons construit les images pour les microservices "pas-ss-0" et "Build_model", en utilisant Dockerfile. Le fichier Dockerfile associé à "pas-ss-0" permet de construire une image avec l'outil "stress-ng", ce dernier permet de stresser le **CPU** et le forcer à être chargé par une valeur personnalisée. D'un autre coté, le fichier Dockerfile associé à "Build_model" permet de construire une image avec toutes les bibliothèques nécessaires pour entraîner le modèle **LSTM**, telles que "Keras", "TensorFlow" et "Numpy". Les images construites à partir de ces deux fichiers Dockerfile, sont uploadé dans le noeud worker, afin qu'elle soit utilisées, par la suite, par les microservices "pas-ss-0" et "Build_model", respectivement. Finalement, nous avons utilisé la bibliothèque "matplotlib" pour dessiner les figures des tests et résultats.

7.2.3 Procédure de Déploiement du Système de Mise à l'Échelle Automatique et Prédictive

Activation du Kubernetes Metrics-Server :

Il correspond à un ensemble de Pods qui vont être exécutés sur le namespace "kube-system", et qui permet de collecter des statistiques sur l'ensemble de Pods exécutant sur le namespace "default", tel que utilisation du CPU, Mémoire.

Les étapes nécessaires pour l'installation de Metrics-Server :

1. `kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.4.1/components.yaml`
2. `kubectl edit deployments.apps -n kube-system metrics-server :`
 - Ajouter "`- --kubelet-insecure-tls=true`" au-dessous de "`- --secure-port=4443`"
 - Ajouter "`- --metric-resolution=5s`" au-dessous de "`- --kubelet-insecure-tls=true`"
 - **5s** représente l'intervalle de monitoring
 - sauvegarder avec "`echap`", "`:wp!`"
3. Essaye la commande "`kubectl top Pods`" : pour voir l'utilisation des ressources (CPU,

Mémoire)

4. Essaye la commande "watch kubectl top Pods" : pour voir l'utilisation des ressources (CPU, Mémoire) en temps réel

Déploiement du Pod sur lequel on va appliquer le mécanisme de mise à l'échelle automatique prédictive : (pas-ss-0)

1. Comme le système MEAP va exercer un stress sur le CPU, on doit construire un container, qui va être exécuté sur un Pod (pas-ss-0), et qui contient le programme "stress-ng" déjà installé. Pour ça, on a créé un fichier dockerfile contenant les packages nécessaires, dans le répertoire : "/home/master/pfe/pas/dockerfiles/server"
2. Exécuter la commande : "sudo docker build -t server:pfe .", il va builder une image "server:pfe" à partir du fichier "dockerfile" existant dans le répertoire courant ".."
3. Sauvegarder l'image créée : "sudo docker save -o server_pfe.tar server:pfe"
4. Copier l'image dans la machine worker : "scp server_pfe.tar worker@10.0.2.5: /home/worker/server/"
5. Comme le cluster Kubernetes installé utilise "containerd" comme gestionnaire de container, on doit uploader l'image à "containerd" dans le worker : "sudo ctr -a /var/run/containerd/containerd.sock -namespace k8s.io image import server_pfe.tar" _ pour voir c'est quoi le container-runtime (docker, containerd), exécuter la commande : "kubectl get nodes -o wide"
6. Dans le Master, aller au répertoire : "/home/master/pfe/pas/kube_template/server"
7. Exécuter la commande : "kubectl apply -f pfe_ss.yaml" _ cette commande va créer un déploiement qui va assurer le fonctionnement du Pod "pas-ss-0", et qui par la suite assure la mise à l'échelle de ce Pod, c'est à dire l'ajout ou la suppression d'un replicas.

Installation de HELM :

Comme un déploiement d'une application sur Kubernetes consiste à installer plusieurs composants séparés, HELM permet, via une seule commande d'installer une application, ce qui est par exemple l'équivalent de "apt-get" sur Linux.

_ Pour installer Helm, nous référons à : "<https://helm.sh/fr/docs/intro/install/>"

Déploiement de influxdb :

1. Aller au répertoire : "/home/master/pfe/pas/kube_template/influxdb"

2. Le répertoire influxdb représente HELM chart
3. Dans le répertoire : "/home/master/pfe/pas/kube_template/influxdb/volumes", il y a 2 fichiers : influxdb-pv.yaml, influxdb-pvc.yaml. Le premier consiste à créer un volume ou espace disque dans : "/mnt/influxdb" dans la machine : "worker". Le 2ème consiste à mapper le Pod influxdb à ce volume
4. Installer le persistent volume via : "kubectl apply -f influxdb-pv.yaml"
5. Installer le persistent volume claim via : "kubectl apply -f influxdb-pvc.yaml"
6. Installer influxdb chart via la commande : "helm install influxdb influxdb"
7. Vérifier si l'installation est correcte : "watch kubectl get all"

Déploiement de Grafana :

1. Aller au répertoire : "/home/master/pfe/pas/kube_template/grafana"
2. Dans le répertoire : "/home/master/pfe/pas/kube_template/grafana", il y a 2 fichiers : grafana-pv.yaml, grafana-pvc.yaml. Le premier consiste à créer un volume ou espace disque dans : "/mnt/grafana" dans la machine : "worker". Le 2ème consiste à mapper le Pod grafana au ce volume
3. Installer le persistent volume via : "kubectl apply -f grafana-pv.yaml"
4. Installer le persistent volume claim via : "kubectl apply -f grafana-pvc.yaml"
5. Déployer grafana via : "kubectl apply -f grafana.yaml"
6. Vérifier si l'installation est correcte : "watch kubectl get all"

Installation des différents packages pour Python3 :

```
— sudo apt install python3-pip
— pip3 install numpy
— pip3 install pandas
— pip3 install sklearn
— pip3 install tensorflow
— pip3 install keras
— pip3 install influxdb
— pip3 install paramiko
— pip3 install matplotlib
— pip3 install cpu_load_generator
```

influxdb CLI :

1. Pour récupérer le nom de influxdb Pod, exécuter la commande : "kubectl get all"
2. Aller à influxdb CLI via : "kubectl exec -it influxdb-0 bash"
3. Créer la base de données "pas" via : "create database pas"
4. Afficher les bases de données via : "show databases"
5. Utiliser la base de données "pas" via : "use pas"
6. Afficher les measurements de "pas" via : "show measurements"

Récupération de l'adresse IP du service influxdb :

l'adresse IP du serveur influxdb correspond à l'adresse du service : "influxdb", qu'on peut la trouver via : "kubectl get all"

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/influxdb	ClusterIP	10.111.219.45	none	8086/TCP,8088/TCP	3d19h

Chargement du measurement "cpu_usage_training_model" :

1. Modifier l'adresse IP du serveur influxdb : "10.111.219.45"
2. Exécuter la commande "python3 load_cpu_usage_training_model.py" (pour lancer le script)
3. Ce script va insérer toutes les données d'utilisation de CPU, à partir du fichier CSV : "cpu_usage_training_model.csv" vers le measurement : "cpu_usage_training_model"
4. Connecter au **CLI** influxdb
5. Utiliser la base de données "pas" via : "use pas"
6. Afficher les measurements de "pas" via : "show measurements"
7. Afficher le contenu de la measurements "cpu_usage_training_model" via : "select * from cpu_usage_training_model"

Construction du build_model image :

build_model permet de générer périodiquement le modèle : "lstm_model.h5", ce qui consomme des ressources s'il sera exécuter sur le "master" node, de ce fait, il est préférable de l'exécuter sur le worker dans un Pod orchestré par Kubernetes.

1. Aller dans le répertoire : "/mnt/bm" dans la machine worker
2. Dedans il y a les 4 fichiers : build_model.py, dockerfile, lstm_model.h5, requirements.txt.

3. Dans le worker, exécuter la commande : "sudo docker build -t bm:pfe ."
- cette commande va créer une image : "bm:pfe"
4. Sauvegarder l'image créée : "sudo docker save -o bm_pfe.tar bm:pfe"
5. Importer l'image à "containerd" via : "sudo ctr -a /var/run/containerd/containerd.sock
-namespace k8s.io image import bm_pfe.tar"
6. Notes bien que le container qui va être créé à partir de cette image, il va lancer le script build_model.py comme point d'entrer. Ce script génère le modèle : "lstm_model.h5" dans un volume, qui va être créé par la suite dans la machine worker "/mnt/bm". Ensuite, il envoie automatiquement "lstm_model.h5" au répertoire : "/home/master/pfe/pas" dans la machine "master"
7. Installation du persistent volume via :
 - "cd /home/master/pfe/pas/kube_template/build_model"
 - "kubectl apply -f bm-pv.yaml"
8. Installation du persistent volume claim via :
 - cd /home/master/pfe/pas/kube_template/build_model
 - kubectl apply -f bm-pvc.yaml
9. Installation du build model deployment : kubectl apply -f bm_ss.yaml

Déploiement de build_model Pod :

1. Aller au répertoire : "/home/master/pfe/pas/kube_template/build_model"
2. Installer le persistent volume via : "kubectl apply -f bm-pv.yaml"
3. Installer le persistent volume claim via : "kubectl apply -f bm-pvc.yaml"
4. Installer build_model via : "kubectl apply -f bm_ss.yaml"
5. Vérifier l'installation via : "watch kubectl get all"
6. Vérifier si le script "build_model.py" est en train de créer du modèle : "lstm_model.h5" et l'envoyer au master via : "kubectl logs -follow bm-ss-0"

Vérification de la plateforme :

Avant de lancer le système "MEAP", les 4 Pods (bm-ss-0, grafana-xxx, influxdb-0, pas-ss-0) doivent avoir l'état : "running" :

NAME	READY	STATUS	RESTARTS	AGE
Pod/bm-ss-0	1/1	Running	1	20h
Pod/grafana-5bdbb749-kddbw	1/1	Running	2	2d4h
Pod/influxdb-0	1/1	Running	2	2d4h
Pod/pas-ss-0	1/1	Running	2	2d4h

Lancement du script : "monitor_cpu.py" , "autoscaler.py" :

1. Exécuter la commande : "python3 monitor_cpu.py"
 - Modifier l'adresse IP du serveur influxdb : "10.111.219.45"
 - Ce script va connecter à influxdb, pour collecter les données à partir de : "cpu_usage_training_model"
 - Ensuite, via ces données, ce script va exercer un stress sur le CPU du Pod : "pas-ss-0",
 - Ensuite, il monitore le CPU via : "kubectl top Pods | grep pas-ss-0"
 - Le résultat du monitoring va être sauvegardée dans le measurement : "monitored_cpu_usage" dans "influxdb"
2. le Pod : "bm-ss-0" est en train de créer le modèle : "lstm_model.h5" périodiquement et le met dans le répertoire : "/home/master/pfe/pas"
3. Exécuter la commande : python3 autoscaler.py
 - Modifier l'adresse IP du serveur influxdb : "10.111.219.45"
 - Ce script va connecter à influxdb, pour collecter les données à partir de : "monitored_cpu_usage"
 - Il prend aussi comme entrée, le modèle entraîné : "lstm_model.h5"
 - à partir des données collectées, il va construire les vecteurs d'entrée du modèle sauvegardé lstm : "lstm_model.h5"
 - Ensuite, via le modèle sauvegardé et le vecteur d'entrée, il produit le vecteur prédit : **yhat**
 - Il teste selon le seuil de mise à l'échelle automatique : "85%", si une valeur dans yhat le dépasse, il déclenche la mise à l'échelle automatique via : "kubectl scale sts pas-ss -replicas="nombre_de_replicas"
 - Il sauvegarde les 2 vecteurs : **input_x**, **yhat** dans influxdb dans les 2 measurements : "cpu_usage_current", "cpu_usage_predicted"

Lancement de Grafana :

1. Ajouter une source de données vers la base de données influxdb : "pas" : "10.111.219.45"
2. Aller à l'URL : <http://10.111.219.45:8086>
3. Connecter via : database: "pas", user: "admin", pw: "admin"
4. Importer le Dashboard déjà enregistré (fichier json)
5. Le Dashboard va afficher dans une figure, l'utilisation actuelle et prédictive, qui sont dans les 2 measurements "cpu_usage_current", "cpu_usage_predicted" dans "influxdb", et qui sont remplis au fur et à mesure en exécutant le script "autoscaler.py"

7.3 Résultats

7.3.1 La méthode Root Mean Square Error (RMSE)

Pour mesurer la justesse de prédiction de **LSTM**, après la phase d'entraînement, on réalise une phase de test pendant laquelle on mesure la différence entre les observations et la prédiction. Il existe différentes métriques de mesure et assez souvent le **RMSE** :

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(\frac{\hat{y}_i - y_i}{n} \right)^2} \quad (7.1)$$

7.3.2 Évaluation du Modèle LSTM

Dans cette section, on va commencer par évaluer le modèle **LSTM** Encodeur-Décodeur qu'on a implémenté pour prédire l'utilisation de **CPU**. Pour réaliser ça, on va recourir à la méthode d'estimation de la précision de la prédiction **RMSE**. Nous rappelons que, dans notre travail, nous avons considéré le **RMSE** par période ou TimeSteps dans lequel on calcule la précision de la prédiction période par période, d'un autre côté le **RMSE** globale consiste à calculer la précision de la prédiction entre une séquence actuelle et prédictive à la fois.

Une fois le modèle prédictif **LSTM** sera évalué, on va présenter quelques résultats d'évaluation de l'approche **MEAP**, en montrant la minimisation de consommation de **CPU**.

7.3.2.1 Le RMSE du modèle LSTM en Changeant la Taille du Batch d'Entrainement (Batch Size)

La figure 7.2 représente l'évolution de "RMSE" en changeant la taille du "batch_size" (batch d'entraînement). On remarque que plus la taille du batch d'entraînement est élevée, plus la prédiction devient mauvaise (proportionnellement), on remarque aussi que la meilleure valeur du "batch size" parmi les valeurs que nous avons essayées est "32", car elle correspond à la plus

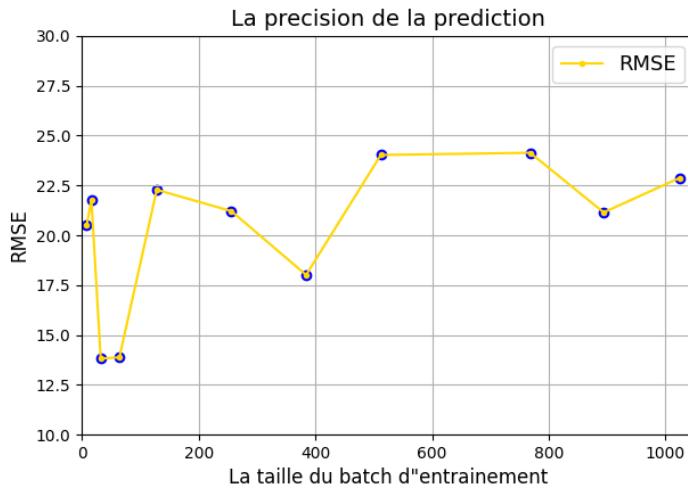


FIGURE 7.2 – RMSE Globale pour Chaque Valeur de Batch size

petite valeur du "RMSE". Sachant que jusqu'ici nous avons choisi les valeurs d'époque d'entraînement et de la taille du vecteur "h" d'une manière spéculative [epoch = 20, la taille du vecteur "h" = 200].

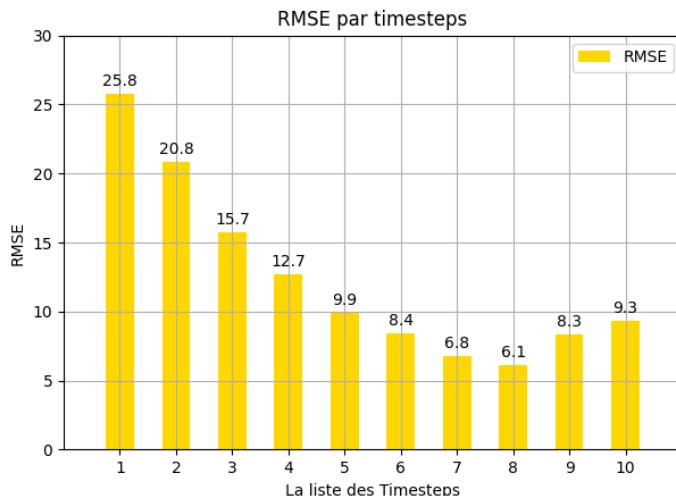


FIGURE 7.3 – RMSE par TimeSteps avec la Meilleur Batch Size

La figure 7.3, représente l'évolution de RMSE par période en utilisant le batch size "32" correspondant au meilleur RMSE trouvé ci-dessus. On remarque que la précision de prédiction s'améliore pour les cellules LSTM les plus loin, contrairement à celles les plus proches, ce qui signifie que la partie long terme du LSTM performe mieux que la partie short terme.

7.3.2.2 Le RMSE du modèle LSTM en changeant le Nombre d'Époques d'entraînement (epoch) et en Utilisant le Meilleur Batch Size Trouvé

La figure 7.4 représente l'évolution de "RMSE" en changeant la taille d'époque d'entraînement et en utilisant la valeur du batch d'entraînement correspondante au meilleur RMSE "32".

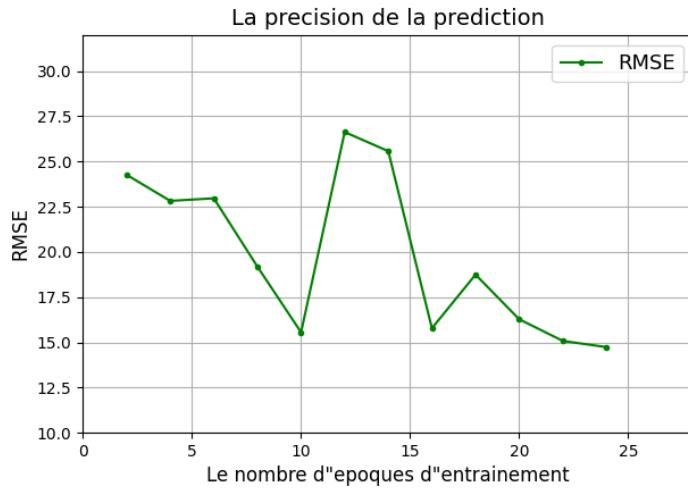


FIGURE 7.4 – RMSE Globale Pour Chaque Valeur d'époque

Comme montre cette figure, les valeurs d'époque très petites ne donnent pas de bonnes prédictions car le RMSE associé est très élevée, on remarque aussi que la meilleure valeur de la taille d'époque parmi les valeurs que nous avons essayées est "24", car elle correspond à la plus petite valeur du "RMSE".

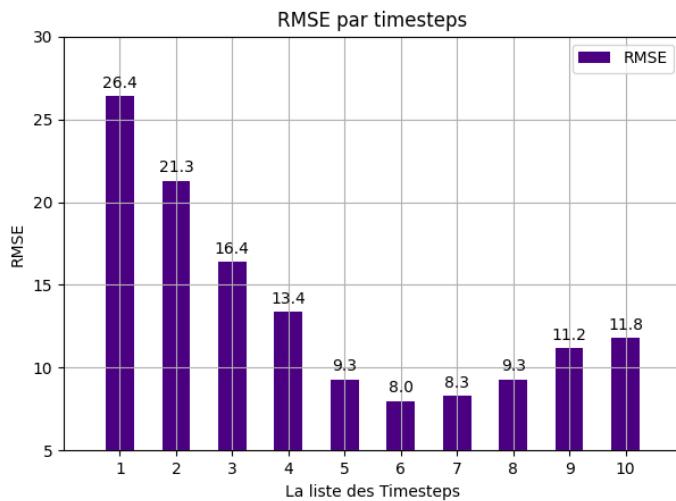


FIGURE 7.5 – RMSE par TimeSteps avec le meilleur nombre d'époques

Sachant que jusqu'ici, nous avons choisi la valeur de la taille du vecteur "h" de manière spé-

culative [taille du vecteur "h" = 200].

La figure 7.5, représente l'évolution de RMSE par période en utilisant le batch size "32" et l'époque "24" correspondants au meilleurs RMSE trouvés ci-dessus. On remarque, similairement au résultat précédent, que la précision de prédiction s'améliore pour les cellules LSTM les plus loin, contrairement à celles les plus proches, ce qui signifie que la partie long terme du LSTM performe mieux que la partie short terme.

7.3.2.3 Le RMSE du Modèle LSTM en Changeant la Taille du Batch d'Entraînement (Batch Size) en Utilisant le Meilleur Nombre d'époques Trouvé

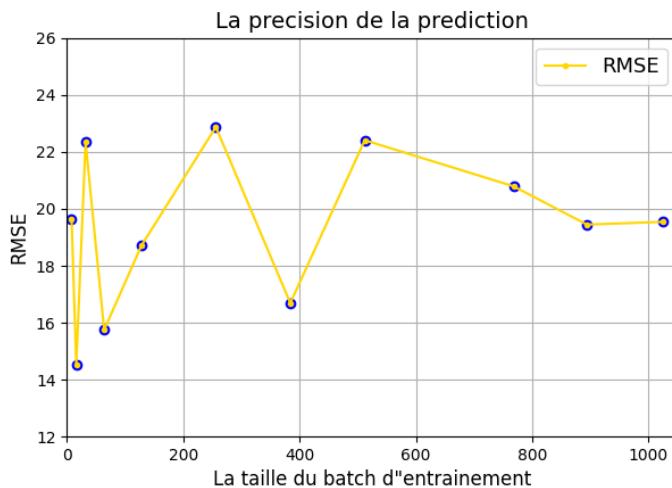


FIGURE 7.6 – RMSE Globale pour Chaque Valeur de Batch Size avec le Meilleur Nombre d'époques Trouvé

La figure 7.6 représente l'évolution de "RMSE" en changeant la taille du "batch_size" (batch d'entraînement) en utilisant la taille d'époques (24) correspondante au meilleur RMSE trouvé sur l'expérience précédente. On remarque qu'il existe une autre valeur pour le "batch_size" qui nous donne une valeur plus petite de "RMSE". À partir de ce résultat, on peut prendre les paramètres : "Batch_size : 16, epoch = 24", sachant que dans cette expérience nous avons aussi choisi la valeur de la taille du vecteur "h" de manière spéculative [taille du vecteur "h" = 200].

La figure 7.7, représente l'évolution de RMSE par période en utilisant le batch size "16" et l'époque "24" correspondants au meilleur RMSE associé à la figure 7.6. On remarque, similairement au résultat précédent, que la précision de prédiction s'améliore pour les cellules LSTM les plus loin, contrairement à celles les plus proches, ce qui signifie que la partie long terme du LSTM performe mieux que la partie short terme.

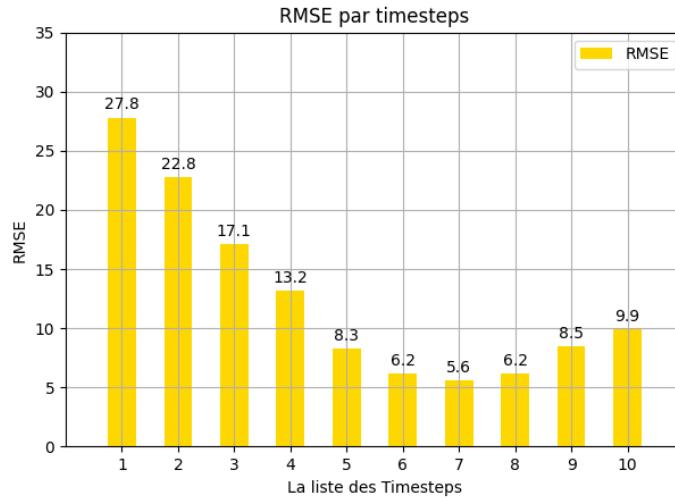


FIGURE 7.7 – RMSE par TimeSteps avec la Meilleur Batch Size

7.3.2.4 Le RMSE du Modèle **LSTM** en Changeant la Taille du Vecteur 'H' en Utilisant les Meilleures Valeurs d'époques et du Batch Size Trouvées

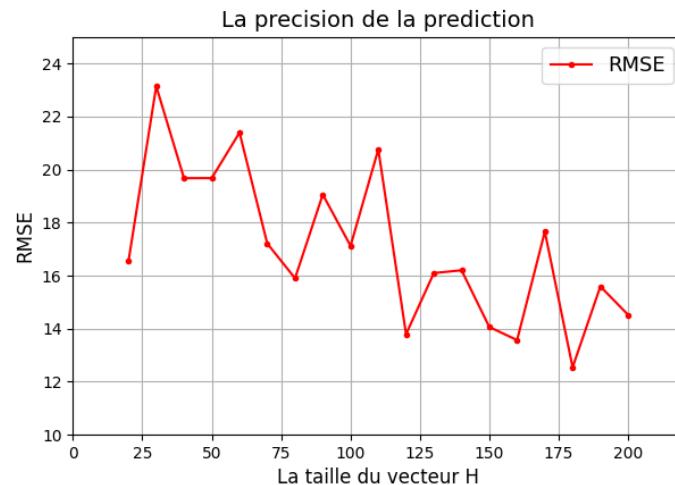


FIGURE 7.8 – RMSE Globale pour Chaque Taille du Vecteur h

La figure 7.8 représente l'évolution de "RMSE" en changeant la taille du vecteur "h". On remarque que plus la taille du vecteur "h" est élevée, plus la prédiction devient mieux (proportionnellement), on remarque aussi que la meilleure valeur du vecteur "h" parmi les valeurs que nous avons essayées est "180" Car elle correspond à la plus petite valeur du "RMSE". Cela est expliqué par le fait que la taille du vecteur "h" représente le nombre de neurones à concaténer avec ceux de l'entrée "X" dans chaque cellule **LSTM**, et l'augmentation du nombre de neurones

implique l'augmentation du nombre de poids "W", ce qui augmente la possibilité de détecter plus de caractéristiques dans chaque séquence.

La figure 7.9, représente l'évolution de RMSE par période en utilisant le batch size "16" et l'époque "24" et la taille du vecteur "h" : "180" correspondants au meilleurs RMSE correspondant à la figure 7.8. On remarque, contrairement au résultats précédents, que la précision de prédiction se dégrade pour les cellules LSTM les plus loins, contrairement à celles les plus proches, ce qui signifie que la partie short terme du LSTM performe mieux que la partie long terme, en changeant la taille du vecteur "h".

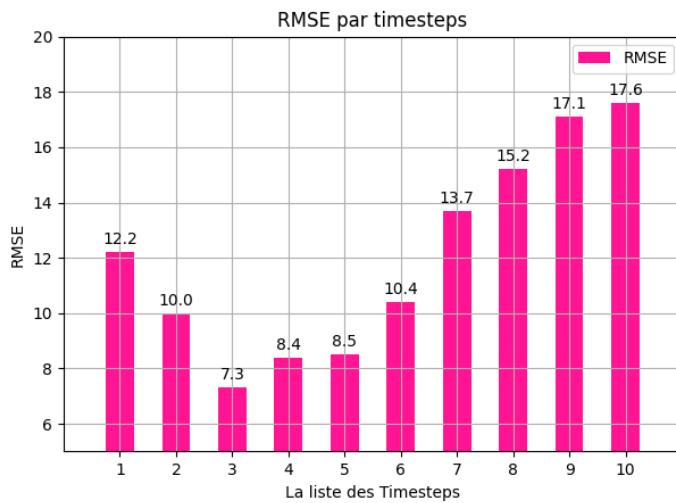


FIGURE 7.9 – RMSE par TimeSteps avec la meilleure taille du vecteur h

7.3.2.5 Le RMSE du Modèle LSTM en Changeant la Fenêtre de Prédiction (TimeSteps)

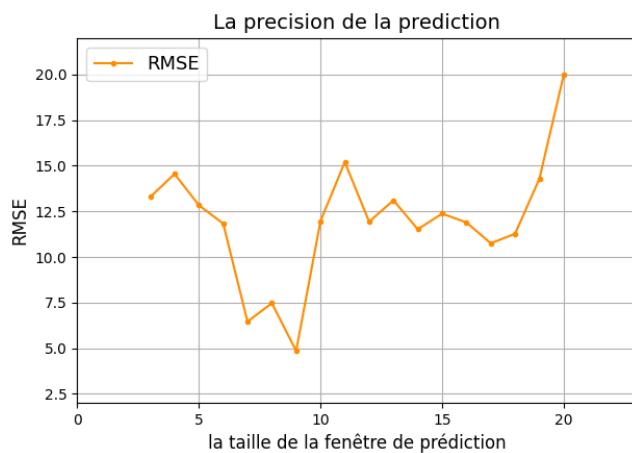


FIGURE 7.10 – RMSE Globale pour Chaque Taille de la Fenêtre de Prédiction (TimeSteps)

Comme notre modèle a pour but de prendre une séquence de consommation de **CPU** et prédire la prochaine séquence, la figure 7.10 représente la précision de prédiction **RMSE** en changeant la taille de la séquence à prédire (TimeSteps), et en utilisant les paramètres trouvés dans les résultats précédents (`batch_size = 16`, `epoch = 24`, vecteur `h = 180`). On remarque similairement au résultat trouvé dans la figure 7.9, la précision de prédiction se dégrade pour les cellule long terme. De ce fait, la valeur de TimeSteps associé au meilleur **RMSE**, correspond à "9". Cela est expliqué par le choix de la taille du vecteur "`h`" : "180", qui a impacté le choix de la taille de la séquence de prédiction "TimeSteps".

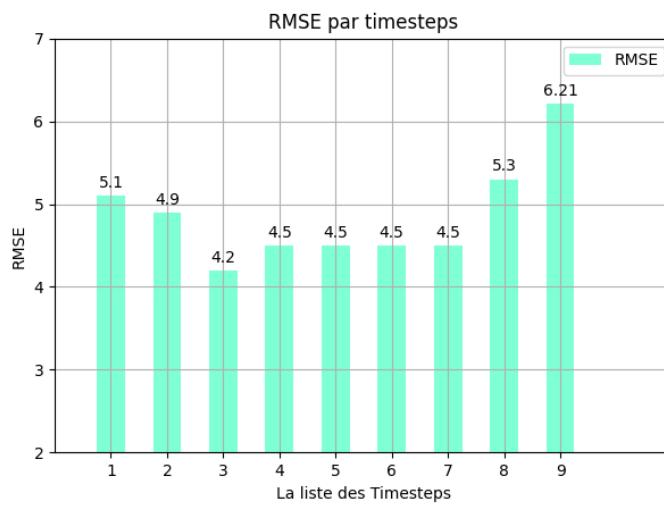


FIGURE 7.11 – **RMSE** par TimeSteps avec la Meilleur Taille de la Fenêtre de Prédiction

La figure 7.11 représente l'évolution de **RMSE** par période en utilisant le batch size "16" et l'époque "24" et la taille du vecteur "h" : "180" et TimeSteps :"9" correspondants au meilleurs **RMSE** correspondant à la figure 7.10. On remarque, contrairement à tous les résultats précédents, que la précision de prédiction est plus ou moins stable, ce qui signifie que les deux parties du **LSTM** : long et short performent bien.

7.3.2.6 Le **RMSE** du Modèle **LSTM** en Changeant le taux d'apprentissage α (learning rate)

La figure 7.12 représente l'évolution de **RMSE** en changeant la valeur du taux d'apprentissage α (Learning Rate). On remarque que la meilleure valeur du **RMSE** correspond à la valeur de α : "0.01". Cela peut être expliqué par le fait que, attribuer des valeurs plus petites à α dégrade la prédiction, car le processus d'entraînement devient très long. D'un autre coté, attribuer des valeurs plus grandes à α , dégrade aussi la prédiction, car la fonction "loss" ne converge pas.

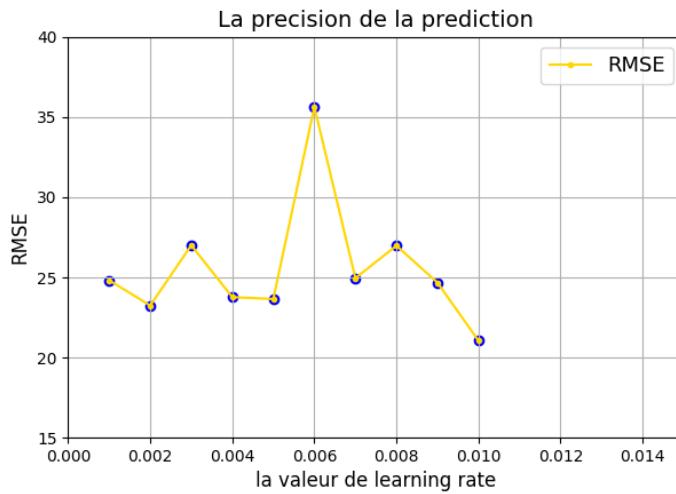


FIGURE 7.12 – RMSE Globale en Changeant la Valeur du taux d'apprentissage (Learning Rate α)

7.3.3 Évaluation de l'approche de Mise à l'Échelle Automatique et Prédictive MEAP

7.3.3.1 Comparaison de la consommation de CPU en utilisant l'approche de mise à l'échelle automatique et prédictive et l'approche de mise à l'échelle automatique non prédictive

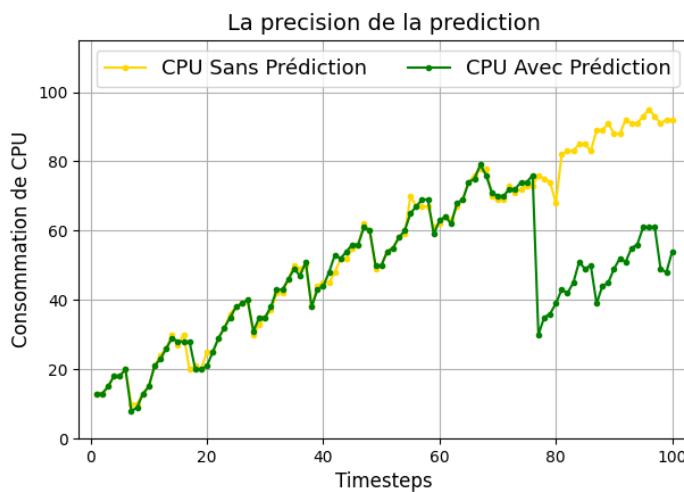


FIGURE 7.13 – Comparaison de la consommation de CPU en utilisant l'approche de mise à l'échelle automatique et prédictive et l'approche de mise à l'échelle automatique non prédictive

La figure 7.13 représente la consommation de CPU avec et sans prédiction, en utilisant le modèle LSTM déjà entraîné avec les meilleurs paramètres trouvés précédemment (Batch size = 16, Epoch = 24, Vecteur "h" = 180, TimeSteps = "10"). Il est à noter que, dans le résultat

précédent nous avons trouvé le meilleur TimeSteps = "9", cependant, le TimeSteps = "10" donne aussi des résultats proches de ceux où TimeSteps = "9", de ce fait nous considérons le TimeSteps = "10".

On peut bien remarquer, à partir de la figure 7.13, que l'activation de la prédiction dans la ligne verte, améliore beaucoup la consommation de CPU, en comparant avec celle où la prédiction est désactivée. De plus, on peut remarquer que la consommation de CPU en activant la prédiction n'atteint jamais le seuil fixé "85%".

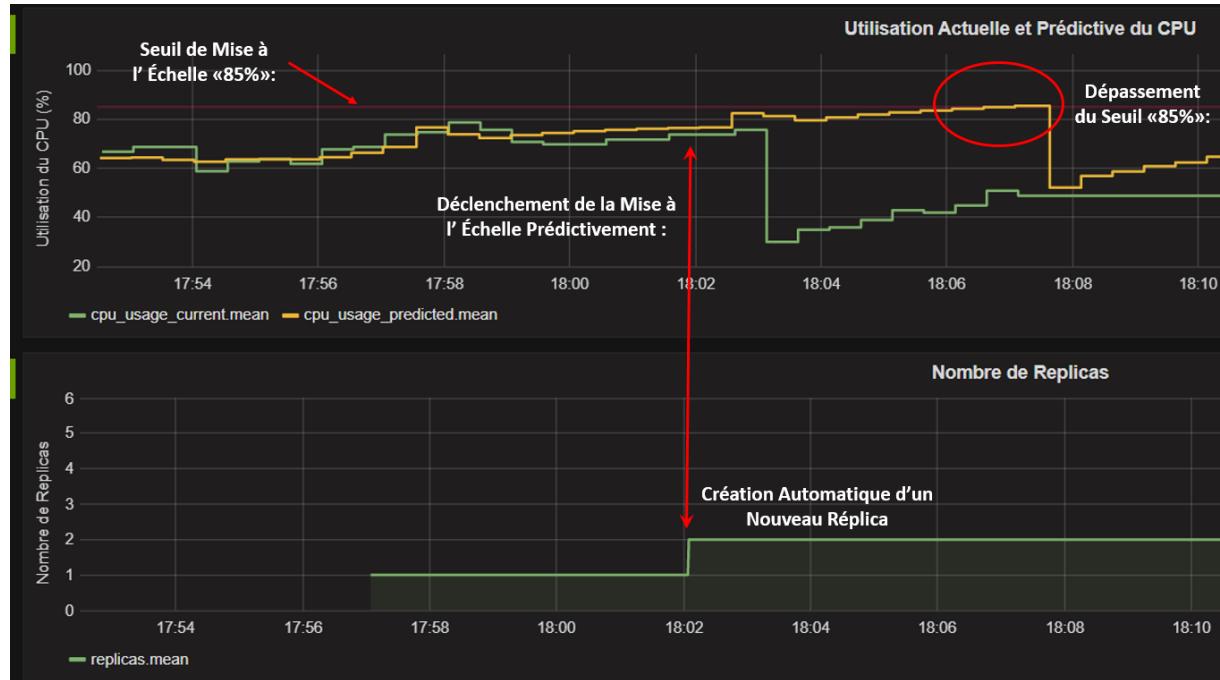


FIGURE 7.14 – Démonstration de MEAP en Utilisant "Grafana"

La figure 7.14 montre la philosophie derrière notre approche proposée MEAP, dans laquelle on démontre l'évolution de la consommation de CPU ainsi que le nombre de réplicas du microservice "pas-ss-0". On peut bien remarquer que MEAP proactivement met à l'échelle le microservices "pas-ss-0", ce qui mène à minimiser la consommation du CPU, où l'utilisation du CPU n'atteint jamais le seuil de "85%".

7.3.3.2 Maximum Consommation de CPU en Variant la Valeur du Seuil de Mise à l'Échelle

Finalement, la figure 7.15 représente les valeurs maximales de consommation de CPU en variant la valeur de seuil de mise à l'échelle. On remarque plus on augmente la valeur du seuil, plus la consommation de CPU augmente, ce qui impacte les performances. De ce fait, nous avons choisi la valeur du seuil de mise à l'échelle "85%". Cependant, il existe un nombre infini de valeurs

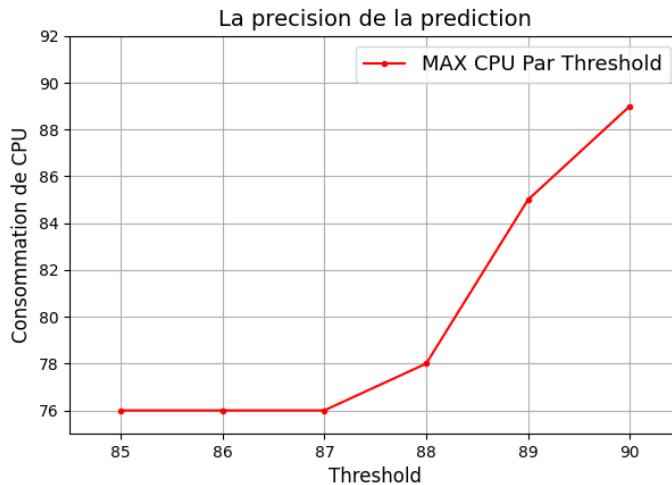


FIGURE 7.15 – Maximum consommation de **CPU** en variant la valeur du seuil de mise à l'échelle qu'on doit tester, ce qui motive à laisser cette question ouverte pour qu'elle soit traitée comme perspective.

7.4 Conclusion

Dans ce chapitre, nous avons évalué notre approche proposée **MEAP**. Nous avons commencé par une présentation de la plateforme de test, en expliquant le rôle de chaque module ainsi que son fonctionnement d'une manière technique. Ensuite, nous avons présenté les technologies et outils utilisés pour implémenter et déployer l'approche **MEAP**, ainsi qu'une procédure de déploiement de la plateforme de test. Après, nous avons présenté des résultats sur : 1) l'évaluation du modèle de prédiction **LSTM**, et 2) l'évaluation de l'approche **MEAP**. Les résultats trouvés montrent l'efficacité de **MEAP** à améliorer l'utilisation des ressources (**CPU**).

Dans le chapitre suivant, on va conclure ce travail et on donne quelques perspectives pour les travaux futurs.

Chapitre 8

Conclusion Générale et Perspectives

*L*a mise à l'échelle automatique et prédictive des ressources aide les fournisseurs de services cloud exploitant des centres de données modernes à prendre en charge un nombre maximal de clients tout en garantissant les exigences de **QoS** des clients conformément aux Accords de Niveau de Service (**SLA**)s et en maintenant le coût d'utilisation des ressources à un faible niveau pour les clients. Ceci en améliorant l'utilisation des ressources (**CPU**, Mémoire) par la minimisation du surchargement des microservices d'une manière proactive.

Dans ce travail, nous avons proposé et évalué le système de mise à l'échelle automatique et prédictive **MEAP** par : 1) étude et familiarisation avec les différentes technologies permettant de mettre en place l'architecture de mise à l'échelle automatique et prédictive, notamment le système de conteneurisation Docker, le système d'orchestration Kubernetes, la méthode de pré-diction **LSTM** et les différents outils permettant de visualiser l'évolution de mise à l'échelle automatique et prédictive, 2) une étude bibliographique et comparative sur les différentes approches proposées dans d'autres travaux de recherches, 3) le déploiement d'un cluster d'orchestration, basé sur Kubernetes **K8s** sur un cluster composé de deux machines virtuelles **MV**, 4) l'activation des modules de monitoring pour permettre une surveillance des microservices, 5) le déploiement d'un système de stress et de monitoring, permettant de stresser le **CPU**, Mémoire d'un microservice déployé sur le cluster Kubernetes, ensuite monitorer la consommation de ses ressources, 6) entraînement du modèle prédictif basé sur le type Encodeur-Décodeur de la méthode **LSTM**, 7) développement d'un système de mise à l'échelle automatique interagissant avec le modèle **LSTM** entraîné et proactivement déclenchant la mise à l'échelle automatique, 8) réalisation de l'entraînement continu du modèle prédictif **LSTM**, 9) démonstration et visualisation de l'évolution d'un scénario de mise à l'échelle automatique et prédictive, en utilisant Grafana et influxdb.

Notre évaluation expérimentale à l'aide de la plateforme de mise à l'échelle automatique et prédictive développée sur le cluster Kubernetes confirme que le système proposé **MEAP** peut

efficacement améliore les performances. En effet, la méthode de prédiction **LSTM** Encodeur-Décodeur a montré son efficacité en terme d'adaptabilité avec des modèles de trafic non réguliers et sa bonne précision de prédiction. De plus, la modularité de notre architecture proposée montre plus de flexibilité et simplicité de considérer l'approche **MEAP**, pour contribuer à améliorer les systèmes d'orchestration, plus spécifiquement la mise à l'échelle automatique et prédictive. D'autre part, ce travail nous a permis de découvrir le système d'orchestration Kubernetes, le système de conteneurisation Docker, ainsi que la méthode de prédiction **LSTM** Encodeur-Décodeur basée sur les réseaux de neurones.

Plusieurs voies de recherche futures s'ouvrent. Dans ce qui suit, nous détaillerons les principaux axes de recherche que nous approfondirons à court, moyen et long termes.

À court terme, en utilisant notre approche proposée **MEAP**, nous envisageons d'évaluer plusieurs stratégies d'optimisation i) en considérant différents types de la méthode **LSTM** et différentes méthodes de prédiction, ii) en améliorant la plateforme de mise à l'échelle automatique et prédictive par l'utilisation d'une **API** dans un microservice pour évaluer l'approche **MEAP**. Cela aiderait certainement les gestionnaires des plateformes d'orchestration à prendre des décisions stratégiques judicieuses et des investissements rentables.

À moyen terme, et comme le système **MEAP** est extensible, nous envisageons de développer davantage notre prototype expérimental **MEAP**, en intégrant d'autres approches proposées dans d'autre travaux de recherches. Cela va nous permettre de comparer **MEAP** avec les autres approches et de tirer partie de leurs avantages en améliorant notre prototype.

Finalement, à long terme, nous envisageons de considérer : i) plusieurs stratégies d'équilibrage de charge entre les différents réplicas d'un même service, ii) le placement des réplicas sur les noeud du cluster Kubernetes (Worker), iii) la mise à l'échelle automatique verticale en ajoutant des ressources **CPU**, Mémoire, iv) une solution hybride dans laquelle on doit trouver un compromis entre la mise à l'échelle automatique horizontale et verticale, v) détermination analytique et pratique du seuil de mise à l'échelle automatique.

Bibliographie

- [1] *À quoi sert la conteneurisation.* [Online]. (consulté le 04/08/2022). URL : <https://www.veritas.com/fr/ch/information-center/containerization>.
- [2] Muhammad ABDULLAH et al. “Burst-aware predictive autoscaling for containerized microservices”. In : *IEEE Transactions on Services Computing* (2020).
- [3] Muhammad ABDULLAH et al. “Learning predictive autoscaling policies for cloud-hosted microservices using trace-driven modeling”. In : *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2019, p. 119-126.
- [4] Muhammad ABDULLAH et al. “Predictive autoscaling of microservices hosted in fog microdata center”. In : *IEEE Systems Journal* 15.1 (2020), p. 1275-1286.
- [5] ANN. [Online]. (consulté le 03/09/2022). URL : [%5Chref%7Bhttps://datascience.eu/fr/apprentissage-automatique/comprendre-les-reseaux-lstm/%7D%7BANN%7D](#).
- [6] ANN architecture. [Online]. (consulté le 03/09/2022). URL : <https://fr.slideshare.net/dabounou/rseaux-de-neurones-rcurrents-et-lstm>.
- [7] API k8S. [Online]. (consulté le 19/08/2022). URL : <https://www.redhat.com/fr/topics/containers/what-is-the-kubernetes-API>.
- [8] Architecture Docker. [Online]. (consulté le 08/08/2022). URL : <https://www.tutorialkart.com/docker/docker-architecture/>.
- [9] Architecture Kubernetes. [Online]. (consulté le 18/08/2022). URL : <https://geekflare.com/fr/kubernetes-architecture/>.
- [10] David BALLA, Csaba SIMON et Markosz MALIOSZ. “Adaptive scaling of Kubernetes pods”. In : *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2020, p. 1-5.

- [11] *Batch Size LSTM*. [Online]. (consulté le 24/09/2022). URL : <https://machinelearningmastery.com/use-different-batch-sizes-training-predicting-python-keras/>.
- [12] *Bidirectional LSTM*. [Online]. (consulté le 23/09/2022). URL : <https://morioh.com/p/df9310bb0277>.
- [13] *BLSTM*. [Online]. (consulté le 15/09/2022). URL : https://www.researchgate.net/figure/Bidirectional-LSTM-model-showing-the-input-and-output-layers-The-red-arrows-represent_fig3_344554659%7D%7BBLSTM%7D.
- [14] *C quoi Docker*. [Online]. (consulté le 04/08/2022). URL : <https://www.journaldunet.com/solutions/cloud-computing/1148473-solomon-hykes-docker-ce-francais-qui-pourrait-revolutionner-l-informatique-mondiale/>.
- [15] *C'est quoi Docker*. [Online]. (consulté le 05/08/2022). URL : <https://www.disko.fr/reflexions/technique/introduction-docker/>.
- [16] Emiliano CASALICCHIO. “A study on performance measures for auto-scaling CPU-intensive containerized applications”. In : *Cluster Computing* 22.3 (2019), p. 995-1006.
- [17] *Cluster k8S*. [Online]. (consulté le 18/08/2022). URL : <https://www.redhat.com/fr/topics/containers/what-is-a-kubernetes-cluster>.
- [18] *CNN*. [Online]. (consulté le 29/08/2022). URL : <https://www.happiestminds.com/insights/convolutional-neural-networks-cnns/#:~:text=Within%5C%20Deep%5C%%5C%2C%5C%20a%5C%20Convolutional,image%5C%20by%5C%20using%5C%20a%5C%20CNN.%7D%7BCNN%7D>.
- [19] *Deep Learning*. [Online]. (consulté le 02/09/2022). URL : <https://www.futura-sciences.com/tech/definitions/intelligence-artificielle-deep-learning-17262/>.
- [20] *Demon Docker*. [Online]. (consulté le 15/08/2022). URL : <https://www.ionos.fr/digitalguide/serveur/configuration/tutoriel-docker-installation-et-premiers-pas/>.
- [21] *Deployment*. [Online]. (consulté le 22/08/2022). URL : <https://kubernetes.io/fr/docs/concepts/workloads/controllers/deployment/>.
- [22] *Deployments*. [Online]. (consulté le 27/08/2022). URL : <https://theithollow.com/2019/01/30/kubernetes-deployments/>.
- [23] *Différents types de réseaux de neurones dans le Deep Learning*. [Online]. (consulté le 03/09/2022). URL : <https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/>.

- [24] *Docker à l'intérieur.* [Online]. (consulté le 14/08/2022). URL : <https://docs.docker.com/get-started/overview/>.
- [25] *Docker initialement.* [Online]. (consulté le 13/08/2022). URL : <https://datascientest.com/docker-guide-complet>.
- [26] *Elément cluster.* [Online]. (consulté le 21/08/2022). URL : <https://www.redhat.com/fr/topics/containers/learning-kubernetes-tutorial#fonctionnement-de-kubernetes>.
- [27] *entraînement_{lstm}.* [Online]. (consulté le 22/09/2022). URL : <https://www.geeksforgeeks.org/lstm-derivation-of-back-propagation-through-time/>.
- [28] *Epoch LSTM.* [Online]. (consulté le 24/09/2022). URL : <https://machinelearningmastery.com/use-different-batch-sizes-training-predicting-python-keras/>.
- [29] *Etcđ.* [Online]. (consulté le 22/08/2022). URL : <https://www.redhat.com/fr/topics/containers/what-is-etcd>.
- [30] *Fenêtre de prédiction LSTM.* [Online]. (consulté le 25/09/2022). URL : <https://machinelearningmastery.com/use-timesteps-lstm-networks-time-series-forecasting/>.
- [31] *Fonction d'Activation "Sigmoid".* [Online]. (consulté le 25/09/2022). URL : <https://www.baeldung.com/cs/sigmoid-vs-tanh-functions>.
- [32] *Fonctionnement de K8S.* [Online]. (consulté le 24/08/2022). URL : <https://www.youtube.com/watch?v=NChhd0ZV4sY>.
- [33] *Forget gate.* [Online]. (consulté le 08/09/2022). URL : <https://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/>.
- [34] Shi HUAXIN et al. “An improved kubernetes scheduling algorithm for deep learning platform”. In : *2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*. IEEE. 2020, p. 113-116.
- [35] *Image Docker.* [Online]. (consulté le 11/08/2022). URL : <https://devopssec.fr/article/fonctionnement-manipulation-images-docker>.
- [36] Li JU, Prashant SINGH et Salman TOOR. “Proactive autoscaling for edge computing systems with kubernetes”. In : *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*. 2021, p. 1-8.
- [37] *K8s.* [Online]. (consulté le 18/08/2022). URL : [%5Chref%7Bhttps://datascientest.com/formation-kubernetes%7D%7BKubernetesw%7D](#).

- [38] Abeer Abdel KHALEQ et Ilkyeon RA. "Agnostic approach for microservices autoscaling in cloud applications". In : *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2019, p. 1411-1415.
- [39] Michael T. KRIEGER et al. "Building an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows". In : *Future Generation Computer Systems* 67 (2017), p. 329-340. ISSN : 0167-739X. DOI : <https://doi.org/10.1016/j.future.2016.02.008>. URL : <https://www.sciencedirect.com/science/article/pii/S0167739X16300218>.
- [40] *kubelet*. [Online]. (consulté le 22/08/2022). URL : <https://www.padok.fr/blog/kubernetes-utilite>.
- [41] *Kubernetes*. [Online]. (consulté le 28/08/2022). URL : %5Chyperref[[\[42\] *Kubernetes*. \[Online\]. \(consulté le 18/08/2022\). URL : <https://www.redhat.com/fr/topics/containers/learning-kubernetes-tutorial#pr%5C%C3%5C%A9sentation>.

\[43\] *L'autoscaling*. \[Online\]. \(consulté le 25/08/2022\). URL : <https://www.densify.com/kubernetes-autoscaling>.

\[44\] *La taille du vecteur "h"*. \[Online\]. \(consulté le 23/09/2022\). URL : <https://medium.com/deep-learning-with-keras/lstm-understanding-the-number-of-parameters-c4e087575756>.

\[45\] *Le but de LSTM*. \[Online\]. \(consulté le 16/09/2022\). URL : <https://openclassrooms.com/fr/courses/5801891-initiez-vous-au-deep-learning/5814656-decouvrez-les-cellules-a-memoire-interne-les-lstm>.

\[46\] *Learning Rate LSTM*. \[Online\]. \(consulté le 22/09/2022\). URL : <https://medium.com/@aidangomez/lets-do-this-f9b699de31d9>.

\[47\] *Learning Rate LSTM*. \[Online\]. \(consulté le 22/09/2022\). URL : <https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/>.

\[48\] *LSTM architecture*. \[Online\]. \(consulté le 07/09/2022\). URL : <https://medium.com/smileinnovation/lstm-intelligence-artificielle-9d302c723eda>.

\[49\] *LSTM empilé*. \[Online\]. \(consulté le 23/09/2022\). URL : <https://machinelearningmastery.com/stacked-long-short-term-memory-networks/>.](https://blog.stephane-robert.info/post/introduction-kubernetes/?fbclid=IwAR2xcL0z6p17rXCicUKEhE4_6Gf-ewWP767mMQeOlgALNIE0df5y8MHff3Y)

- [50] *LSTM Encoder Decoder*. [Online]. (consulté le 20/09/2022). URL : <https://pradeep-dhote9.medium.com/seq2seq-encoder-decoder-lstm-model-1a1c9a43bbac>.
- [51] *Lstm fonctionnement*. [Online]. (consulté le 04/09/2022). URL : https://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm/?fbclid=IwAR2a5nwZGXSal0H0_GfV_pDWo00lWVfluHGeVxJAchEIVXDvlo-gCBH2ro.
- [52] *LSTM_encoder_decoder*. [Online]. (consulté le 20/09/2022). URL : <https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/>.
- [53] *Machine Learning*. [Online]. (consulté le 30/08/2022). URL : <https://www.journaldunet.fr/web-tech/guide-de-l-intelligence-artificielle/1501881-machine-learning/#:~:text=Le%20machine%20learning%20est%5C%20une,de%5C%20r%C3%A9aliser%5C%20des%5C%20t%C3%A9chniques%5C%20complexes..>
- [54] *Machine Learning*. [Online]. (consulté le 01/09/2022). URL : <https://www.talend.com/fr/resources/what-is-machine-learning/>.
- [55] *Many to Many*. [Online]. (consulté le 19/09/2022). URL : <https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/>.
- [56] Nicolas MARIE-MAGDELAINE et Toufik AHMED. “Proactive Autoscaling for Cloud-Native Applications using Machine Learning”. In : *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE. 2020, p. 1-7.
- [57] *Microservices*. [Online]. (consulté le 02/08/2022). URL : <https://www.redhat.com/fr/topics/microservices>.
- [58] *MLP*. Overview. [Online]. URL : <https://blog.octo.com/des-reseaux-de-neurones-pour-generer-des-discours-politiques/>.
- [59] *Mode d'utilisation de LSTM*. [Online]. (consulté le 21/09/2022). URL : <https://datasciencetoday.net/index.php/fr/machine-learning/148-reseaux-neuronaux-recurrents-et-lstm>.
- [60] *Noeud cluster*. [Online]. (consulté le 21/08/2022). URL : <https://www.redhat.com/fr/topics/containers/kubernetes-architecture#que-se-passe-il-dans-un-n%C5%93ud%C2%5C%A0>.
- [61] *Noeud cluster*. [Online]. (consulté le 20/08/2022). URL : <https://www.padok.fr/blog/kubernetes-pods-services>.
- [62] *Orchestration*. [Online]. (consulté le 17/08/2022). URL : <https://www.redhat.com/fr/topics/containers/what-is-container-orchestration>.

- [63] *Orchestration*. [Online]. (consulté le 16/08/2022). URL : <https://www.intel.fr/content/www/fr/fr/cloud-computing/containers.html>.
- [64] Vladimir PODOLSKIY et al. “Forecasting models for self-adaptive cloud applications : A comparative study”. In : *2018 ieee 12th international conference on self-adaptive and self-organizing systems (saso)*. IEEE. 2018, p. 40-49.
- [65] *Problème de dépendance à long terme*. [Online]. (consulté le 05/09/2022). URL : <https://datapeaker.com/big-data/multicolinealidad-detectando-multicolinealidad-con-vif/>.
- [66] Babak Bashari RAD, Harrison John BHATTI et Mohammad AHMADI. “An introduction to docker and analysis of its performance”. In : *International Journal of Computer Science and Network Security (IJCSNS) 17.3* (2017), p. 228.
- [67] *Réplicasat*. [Online]. (consulté le 27/08/2022). URL : <https://blog.engineering.publicissapient.fr/2020/06/30/39809/>.
- [68] *Replicaset*. [Online]. (consulté le 22/08/2022). URL : <https://kubernetes.io/fr/docs/concepts/workloads/controllers/replicaset/>.
- [69] *Retropagationlstm*. [Online]. (consulté le 22/09/2022). URL : <https://datasciencetoday.net/index.php/fr/machine-learning/148-reseaux-neuronaux-recurrents-et-lstm>.
- [70] *RNN introduction*. [Online]. (consulté le 18/09/2022). URL : <https://openclassrooms.com/fr/courses/5801891-initiez-vous-au-deep-learning/5814651-decouvrez-le-fonctionnement-des-reseaux-de-neurones-recurrents>.
- [71] Lluís Mas Ruíz et al. “Autoscaling Pods on an On-Premise Kubernetes Infrastructure QoS-Aware”. In : *IEEE Access* 10 (2022), p. 33083-33094. DOI : [10.1109/ACCESS.2022.3158743](https://doi.org/10.1109/ACCESS.2022.3158743).
- [72] *StatefulSets*. [Online]. (consulté le 27/08/2022). URL : https://kubernetes.io/fr/docs/concepts/_print/#pg-6d72299952c37ca8cc61b416e5bdbcd4.
- [73] Mulugeta Ayalew TAMIRU et al. “An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud”. In : *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2020, p. 17-24.
- [74] Xuxin TANG et al. “Quantifying Cloud Elasticity with Container-Based Autoscaling”. In : *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. 2017, p. 853-860. DOI : [10.1109/DASC-PICom-DataCom-CyberSciTec.2017.143](https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2017.143).

- [75] *Timesteps LSTM*. [Online]. (consulté le 23/09/2022). URL : <https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-019-0775-2/figures/1>.
- [76] Fan-Hsun TSENG et al. “A Lightweight Autoscaling Mechanism for Fog Computing in Industrial Applications”. In : *IEEE Transactions on Industrial Informatics* 14.10 (2018), p. 4529-4537. DOI : [10.1109/TII.2018.2799230](https://doi.org/10.1109/TII.2018.2799230).
- [77] *Type d'autoscaling*. [Online]. (consulté le 25/08/2022). URL : <https://www.densify.com/kubernetes-autoscaling/kubernetes-hpa>.
- [78] *Vecteur h*. [Online]. (consulté le 16/09/2022). URL : <https://tung2389.github.io/coding-note/unitslstm>.
- [79] Soumia ZERTAL. *Cours cloud virtualisation*. Jan. 2020. DOI : [10.13140/RG.2.2.35943.60321](https://doi.org/10.13140/RG.2.2.35943.60321).

Construction des Images : "pas-ss-0" et "build_model"

A.1 Introduction

*O*n montre dans cette annexe, les contenus des deux fichiers "Dockerfile", utilisés pour construire les deux images : "pas-ss-0" dans la Section A.2 et "build_model" dans la Section A.3.

A.2 Le Fichier Dockerfile Utilisé pour construire l'Image du Microservice "pas-ss-0"

```
from ubuntu:latest

RUN apt-get update  apt-get install -y iutils-ping  apt-get install -y net-tools
apt-get install -y stress-ng

CMD [ "/bin/bash", "-ce", "tail -f /dev/null" ]
```

A.3 Le Fichier Dockerfile Utilisé pour construire l'Image du Microservice "build_model"

```
FROM python:3.6-slim

RUN apt-get update
RUN apt-get -y install
python3-dev
build-essential
```

```
iputils-ping
net-tools
vim
WORKDIR /lstm
COPY requirements.txt /lstm/requirements.txt
RUN pip3 install -r requirements.txt -src /usr/local/src
CMD [ "python3", "-u", "buildmodel.py"]
CMD["/bin/bash", "-ce", "tail -f /dev/null"]
```

Le Fichier "requirements.txt" :

```
numpy
pandas
sklearn
tensorflow
keras
influxdb
paramiko
```

Les Fonctions d'Activation Utilisées par LSTM

B.1 Introduction

*O*n montre dans cette annexe, les deux fonctions d'activation utilisées par **LSTM**. Nous commençons par la fonction "Sigmoid" dans la Section **B.2**, ensuite, la fonction "Tanh" dans la Section **B.3**.

B.2 La Fonction d'activation Sigmoïde

La fonction d'activation sigmoïde (également appelée fonction logistique) prend n'importe quelle valeur réelle en entrée et génère une valeur dans la plage $[0, 1]$. Il est calculé comme suit :

$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}.$$

Où x est la valeur de sortie du neurone. La figure **B.1** montre le tracé de la fonction sigmoïde lorsque l'entrée se situe dans la plage $[-10, 10]$, [31] :

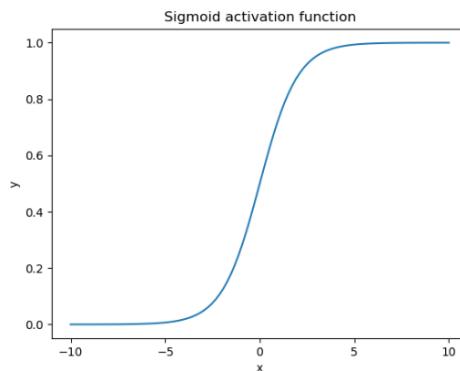


FIGURE B.1 – La Fonction Sigmoïde [31]

B.3 La Fonction d'activation tangente hyperbolique (Tanh)

C'est une fonction d'activation courante dans l'apprentissage en profondeur. Elle est calculé comme suit :

$$\text{Tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = \frac{2}{1 + e^{-2x}} - 1 = 2 \times \text{Sigmoid}(2 \times x) - 1.$$

La fonction tanh est une version décalée et étirée de la sigmoïde. La figure B.2 montre le tracé de la fonction tanh lorsque l'entrée est dans la plage [-10, 10], [31] :

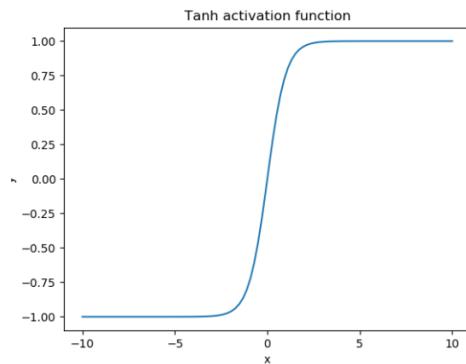


FIGURE B.2 – La Fonction Tanh, [31]