Sûreté de fonctionnement et provisionnement éco-énergétique dans les centres de données virtualisés IaaS

Zeineb Tayachi

École Doctorale Informatique, Télécommunications et Électronique (Paris)

Le Centre d'étudies et de Recherche en Informatique et Communications

École doctorale Sciences et Techniques de l'Ingénieur, Systèmes de Communications, Laboratoire Systèmes de Communications

THÈSE

présentée par : **Zeineb Tayachi**

soutenue le :**09 juillet 2021**

pour obtenir le grade de **DOCTEUR** *conjointement* **du Conservatoire National des Arts et Métiers** et **l'École Nationale d'Ingénieurs de Tunis**

Spécialité : **Informatique**

Sûreté de fonctionnement et provisionnement éco-énergétique dans les centres de données virtualisés IaaS

THÈSE dirigée par :

M. Kamel BARKAOUI	Professeur des universités, Conservatoire national des arts et métiers.
M. Mohamed ESCHEIKH	Maître de conférence, École nationale d'ingénieurs de Tunis (ENIT).

RAPPORTEURS :

M. Djamil AISSANI	Professeur des universités, Université de Bejaia
Mme. Nihal PEKERGIN	Professeur des universités, Université Paris Est Créteil

PRÉSIDENT DU JURY :

M. Karim DJOUANI	Professeur des universités, Université Paris Est Créteil

EXAMINATEURS :

Mme Samia BOUZEFRANE	Professeur des universités, Conservatoire national des arts et métiers

Remerciements

Les travaux présentés dans ce manuscrit ont fait l'objet d'une thèse cotutelle entre le Conservatoire national des arts et métiers (CNAM) et L'École nationale d'ingénieurs de Tunis (ENIT).
Tout d'abord, je tiens à remercier mon directeur de thèse, monsieur Kamel Barkaoui, professeur des universités au département informatique (EPN 5) du Conservatoire national des arts et métiers (CNAM) d'avoir accepté de diriger mes travaux de recherches, pour son accueil et ses qualités scientifiques et humaines.
Je tiens également à exprimer ma plus profonde reconnaissance à mon co-directeur monsieur Mohamed Escheikh, maître de conférence au Laboratoire systèmes de communication à l'École nationale d'ingénieurs de Tunis (ENIT) pour son encadrement exemplaire et ses qualités professionnelles.
Je souhaite aussi remercier vivement les membres de mon jury.
Enfin, je tiens à exprimer ma profonde reconnaissance à mon mari que je remercie pour son aide considérable et tout l'amour qu'il me porte, ma famille et mes amis pour leur soutien et leurs encouragements.

Résumé

Le Cloud computing offre aux utilisateurs l'opportunité d'exploiter des services qui peuvent être des infrastructures, des plateformes, des applications,... Ceci représente un gain de temps et d'argent considérable puisque l'utilisateur n'a besoin ni d'investir dans une infrastructure onéreuse, ni de gérer sa maintenance, de plus il paye juste les ressources utilisées. Afin de prendre en charge des applications à grande échelle et stocker de gros volumes de données, les centres de données ont été largement déployés par des fournisseurs cloud. Cependant, les études ont montré une mauvaise utilisation des ressources qui ne sont pas exploitées pleinement. Les technologies de virtualisation ont permis d'améliorer la situation en déployant des centres de données virtualisés. Ces derniers sont des centres de données où tout ou une partie du matériel (par exemple, serveurs, routeurs, commutateurs et liens) est virtualisé à l'aide d'un logiciel appelé hyperviseur qui divise l'équipement en plusieurs instances virtuelles isolées et indépendantes (comme des machines virtuelles). Toutefois les performances des équipements peuvent être atténuées à cause de plusieurs phénomènes tels que le vieillissement logiciel.

L'objectif de cette thèse est d'évaluer les performances de deux composants clés dans les centres de données qui sont le serveur virtualisé et le commutateur virtuel. Puisque, l'architecture de ces systèmes est complexe, on a opté pour utiliser les formalismes de modélisation avant de mettre en place des solutions pratiques.

Notre première contribution concerne la modélisation et l'évaluation de la performabilité d'un serveur virtualisé qui implémente une politique de gestion de l'énergie et utilise le rajeunissement logiciel comme une technique proactive afin de prévenir les aléas de vieillissement logiciel. Cette modélisation est basée sur une approche modulaire utilisant le SRN (Stochastic Reward Nets) qui décrit les différents états du SVS ainsi que les transitions régissant le passage d'un état à l'autre. L'analyse numérique permet de capturer l'impact de la variation de la charge de travail et le trafic en rafale sur les métriques de performabilité, ce qui permet de bien définir les paramètres du système.

La seconde contribution porte sur l'évaluation des performances d'un commutateur virtuel qui détermine les performances du réseau puisqu'il établit la communication entre des VMs. Le modèle analytique proposé représente l'architecture interne de ce nœud critique avec plusieurs cartes d'interface réseau (représentant des ports) et plusieurs cœurs de processeur (CPU). Chaque CPU sert un ensemble de ports. Le modèle est basé sur les files d'attente avec serveur en vacance et des arrivées groupées. Les résultats numériques montrent l'impact de la taille du groupe et la politique d'acceptation sur les performances du commutateur. Ces résultats peuvent être intéressants lors du dimensionnement des ressources d'un commutateur virtuel.

Mots clés

Cloud Computing, centre de données virtualisé, serveur virtualisé, commutateur virtuel, modélisation, métriques de performance.

Abstract

Cloud computing allows users to exploit services such as infrastructures, platforms, applications, ...This allows a considerable cost and time saving since users do not need buying and managing of equipment. Moreover, they just pay the resources used (pay-as-you go). With the increasing large-scale applications and the need to store huge quantities of data, data centers have been widely deployed. However, studies have shown the under utilization of resources. Therefore, Cloud providers resort to virtualization technologies that are adopted by data center architectures and virtualized data centres have been deployed. A Virtualized Data Center is a data center where some or all of the hardware (e.g, servers, routers, switches, and links) are virtualized by using software called hypervisor that divides the equipment into multiple isolated and independent virtual instances (e.g virtual machines (VMs)). However, equipment performance can be mitigated due to several phenomena such as software aging.
In this thesis, we focus on performance evaluation of two components in the data centers which are the virtualized server and the virtual switch, by using modeling formalisms.
The first contribution concerns performability modeling and analysis of server virtualized systems subject to software aging, software rejuvenation and implements an energy management policy. A modular approach based on SRNs is proposed to investigate dependencies between several server virtualized modules. Numerical analysis shows how workload with bursty nature impacts performability metrics. This can handle decision making related to rejuvenation scheduling algorithms and to select the suitable rejuvenation mechanism.
The second contribution concerns virtual switch (VS) which is considered as key element in data center networks since it achieves the communication between virtual machines. An analytical queueing model with batch arrivals and server vacations is proposed to evaluate VS performance with several network interface cards and several CPU cores. Performance metrics are obtained as a function of two proposed batch acceptance strategies and mean batch size. Numerical results are meaningful when sizing virtual switch resources.

Key Words
Cloud Computing, virtualized data center, virtualized server, virtual switch, modeling, performance metrics.

Table des matières

Liste des figures

Liste des tableaux

Chapitre 1

Introduction Générale

1.1 Contexte et motivations

Dans un contexte caractérisé aujourd'hui par une concurrence ardue, les entreprises en général et les directions des systèmes d'information (DSI) en particulier s'orientent de plus en plus vers des solutions cloud, afin de fructifier leurs investissements clés et à tirer une meilleure valeur ajoutée de leurs systèmes d'information et de leurs processus métier. Cette tendance est d'autant renforcée par l'émergence d'une nouvelle génération de centres de données informatiques les data center green-IT qui constitueront la clé de voûte du cloud computing futur et contribuent substantiellement à performer une efficacité énergétique tout en maintenant une haute performabilité. Les centres de données, grâce à la virtualisation, évoluent aujourd'hui à grande échelle dans le monde entier vers des modèles de déploiement virtualisés de type cloud computing. Ceci permettra d'améliorer non seulement l'agilité et la réactivité de l'entreprise mais aussi de réaliser des économies.
L'avènement du cloud computing (informatique en nuage ou dématérialisée) offre aujourd'hui aux entreprises de réelles opportunités pour une meilleure maîtrise des coûts aussi bien CAPEX (CAPital EXpenditure : dépenses d'investissement) que OPEX (OPerational EXpenditure : dépenses d'exploitation). La virtualisation consiste à découpler le mappage physique entre le logiciel et le matériel. Ceci permet une meilleure utilisation des ressources disponibles (grâce à la virtualisation les capacités matérielles sont fortement optimisées), des coûts réduits, des économies d'énergie, et une meilleure agilité. Afin de répondre aux différents besoins des utilisateurs, il existe plusieurs niveaux et types de virtualisation. On peut citer la virtualisation des serveurs, des systèmes d'exploitation, des postes de travail, des applications, du stockage, et de réseau.
La virtualisation des serveurs ainsi que la virtualisation de réseau sont aujourd'hui les types de technologie clé utilisées pour fournir une infrastructure logicielle de divers services tels que le cloud computing et la gestion de la disponibilité des systèmes.
Le principe de la virtualisation des serveurs consiste à transférer les applications professionnelles, hébergées dans des serveurs physiques, généralement sous-utilisées, vers des serveurs virtuels s'exécutant sur un nombre moins important de machines physiques. La virtualisation de réseau, on l'appelle aussi des fonctions réseau (NFV : Network Functions Virtualization) consiste à virtualiser les services réseau (commutateur, routeur, pare-feu, modules d'équilibrage de charge, etc.) traditionnellement exécutés sur du matériel propriétaire.
Aujourd'hui, les centres de données sont basés sur la technologie de virtualisation afin de bénéficier au mieux des opportunités qu'elle offre comme la fourniture de solutions à haut niveau de disponibilité, l'augmentation du taux d'utilisation des ressources à travers la consolidation et l'amélioration de la flexibilité à fin de faire face à une demande élastique. Cependant pour exploiter au mieux la virtualisation et pour tirer les meilleurs profits des avantages, des défis restent encore à relever. À fin d'assurer une haute disponibilité, les centres de données virtualisés ont investi dans des solutions permettant la gestion des interruptions et les pannes non planifiées des applications

critiques et des restaurations et des reprises rapides sur incidents.
Dans cette thèse, nous sommes intéressés par l'une des causes de pannes de logiciel qui est le vieillissement logiciel. Le vieillissement logiciel se manifeste par une dégradation de la performance du logiciel et par la détérioration du taux de disponibilité des ressources. Ce phénomène devient critique dans les applications et peut causer des catastrophes. Afin de prévenir les dégâts causés par le vieillissement logiciel, des travaux de recherches ont été réalisés. Une approche proactive a été proposée qui consiste à nettoyer et restaurer l'état de l'application en supprimant les erreurs accumulées et libérant les ressources du système. Cette approche est appelée le rajeunissement logiciel qui profite des atouts de la virtualisation qui est la migration. Le rajeunissement logiciel peut affecter le VMM (Virtual Machine Monitor : moniteur de machine virtuelle)et le VM (Virtual Machine : machine virtuelle). Puisque les VMs sont gérées par les VMM, il existe une corrélation entre rajeunissement VMM et rajeunissement VM. Cette corrélation sera détaillée et étudiée dans les prochains chapitres.
D'une autre part, on a cité parmi les avantages de la virtualisation la réduction des coûts CAPEX et OPEX. En effet, la virtualisation utilise moins de serveurs grâce à la consolidation et donc moins d'énergie utilisée et donc moins de pollution. Mais aussi afin d'assurer une haute disponibilité, les administrateurs de centres de données ont mis en place plusieurs serveurs sous-utilisés afin d'assurer une redondance matérielle si une défaillance aura lieu, ceci qui engendra un gaspillage d'énergie. D'après les statistiques, la consommation d'énergie des centres de données représente 1% de la consommation mondiale et ce chiffre ne cesse pas d'augmenter.
Pour diminuer la consommation d'énergie, des études ont été réalisées pour mettre en place des mécanismes permettant de résoudre cette contrainte. La gestion énergétique dans les centres de données a été traitée sur plusieurs volets. Plusieurs recherches se concentrent sur les systèmes de refroidissement en choisissant un emplacement avec les conditions climatiques appropriées qui permet de refroidir les systèmes (air ou eau fraîche), au lieu d'utiliser des systèmes de refroidissement mécaniques. D'autres études ont montré qu'il existe un gaspillage d'énergie côté serveur puisque ce dernier fonctionne à 10-50% de sa capacité la plupart du temps.
Nous avons traité dans ce travail l'inefficacité énergétique dans les centres de données côté serveurs en proposant des politiques de gestion d'énergie.
Pour la première partie de notre thèse, nos défis concernent notamment :

— le traitement du problème du vieillissement logiciel en proposant des solutions proactives telles que le rajeunissement logiciel ;
— trouver les intervalles optimaux pour effectuer les rajeunissements logiciels en tenant compte de la charge de travail ;
— la fourniture de services fiables avec haute disponibilité ;
— faire face aux menaces résultant des fluctuations de charges imprévues, les pannes matérielles et logicielles, et les attaques de réseau ;
— la rationalisation de la consommation de l'énergie afin d'améliorer la maîtrise des coûts ;
— l'adoption de politiques de gestion efficaces.

Dans la deuxiéme partie de notre travail, nous nous intéressons aux équipements utilisés dans les infrastructures réseau courant SDN/NFV plus précisément le commutateur virtuel. C'est un équipement clé dans l'architecture SDN/NFV. Il établit la communication entre des VMs existant sur le même hôte physique ou sur un hôte différent. Il est chargé de la réception, du traitement, de la commutation et de la transmission des paquets circulant dans le réseau. Un commutateur virtuel gére ses ressources avec une grande flexibilité. En effet, plusieurs ressources comme les ports et les cœurs de processeur (CPU) peuvent être allouées ou dés-allouées de manière dynamique selon la charge de travail. Ceci permet une utilisation optimale des ressources et aussi une réduction de la consommation d'énergie. Par exemple, si un commutateur virtuel fait face à une charge de travail excessive, l'hyperviseur peut allouer des CPU supplémentaires pour faire face à l'augmentation de la charge. Dans le cas inverse, une fraction de ressources peut être dés-allouer,

ce qui permet une réduction de la consommation électrique.
Afin d'évaluer les performance du commutateur virtuel aux fluctuations imprédictibles du trafic , nous avons modéliser son architecture sous certaines hypothèses par un modèle d'attente.

1.2 Contributions de cette thèse

Le Cloud Computing c'est le paradigme qui offre des services à la demande via internet aux utilisateurs grâce à la technologie de virtualisation. Il existe trois modèles de services : IaaS-Infrastructure-as-a-Service, PaaS-Plateform-as-a-Service, SaaS-Software as a Service.
Dans cette thèse, nous sommes intéressés par le niveau IaaS. C'est le service de plus bas niveau qui délivre des ressources d'infrastructure telles que des serveurs, des réseaux, de l'espace de stockage... Ces ressources sont gérées et hébergées dans des centres de données. Un contrat sera établi entre le fournisseur d'IaaS et son client, où le client loue une infrastructure informatique pour exécuter ses applications et paie ce qu'il utilise. À son tour, le fournisseur d'IaaS s'engage à garantir une disponibilité élevée et une meilleure qualité de service. Cependant, avec la complexité croissante des systèmes, des architectures des réseaux de communication et des services, le Cloud Computing est toujours vulnérable à un grand nombre de problèmes tels que les pannes des systèmes, le gaspillage énergétique et la sûreté de fonctionnement.
Afin de relever ces défis, il est important d'identifier les contraintes en termes de ressources énergétiques, de traitement et des exigences en termes de qualité de service et de scalabilité. La modélisation des systèmes représente aujourd'hui un grand intérêt pour les fournisseurs d'IaaS permettant la prédiction et l'évaluation des performances des systèmes. Cette thèse présente deux contributions.

Le première partie de notre problématique, sur lequel nous nous focalisons dans ce travail, concerne la modélisation et l'évaluation de la performabilité d'un système de serveurs virtualisés (SVS) basée sur une approche modulaire utilisant le formalisme SRN (Stochastic Reward Nets). Ce dernier permet de développer des modèles riches et concis grâce à sa présentation détaillée. On décrit les différents états ainsi que les transitions qui peuvent être franchies par des gardes.
Le modèle met en place une technique proactive appelée le rajeunissement logiciel pour prévenir les pannes causées par le vieillissement logiciel. Aussi, une politique de gestion dynamique d'énergie a été integrée à travers les différents états (P-state et C-state) dans le but de minimiser la consommation d'énergie. Pour résumer, nos contributions peuvent être décrites à partir des questions suivantes :

— Comment à partir d'une modélisation formelle, modéliser différentes entités représentant respectivement un système de serveurs virtualisés à savoir : le VMM, les VMs et le PMC (Power Manageable Component) ?

— Comment concevoir et implémenter une politique de gestion dynamique de l'énergie et l'intégrer dans un modèle de disponibilité d'un SVS ?

— Comment définir des métriques de performabilité à partir du modèle du SVS mettant en évidence l'efficacité énergétique, la haute disponibilité et les compromis nécessaires énergie-performance ?

Pour la deuxième partie de notre problématique, nous sommes intéressés à l'évaluation des performances d'un nœud critique dans l'architecture du réseau dans les centres de données. On a modélisé l'architecture du commutateur virtuel composée par plusieurs CPU et des ports E/S par un système de file d'attente. Afin de simplifier la résolution du système, nous adoptons une décomposition du système en un ensemble de files d'attente indépendantes. Chaque file est associée à un seul port. L'interaction entre les files d'attentes servies par le même CPU est représentée par un serveur avec vacance. Le modèle proposé est une généralisation d'un modèle proposé dans

la littérature. Des paramètres des performances ont été analysés en utilisant des méthodes numériques pour résoudre le système.

1.3 Organisation du manuscrit

Ce manuscrit est composé de deux parties, une conclusion générale et s'achève par une bibliographie. La première partie concerne les contextes et l'état de l'art, elle est constituée de deux chapitres :

— Le chapitre 2 introduit les différents domaines dans lesquels portent les contributions de cette thèse. Nous présentons en premier lieu les principaux concepts liés au Cloud Computing et la technologie de virtualisation. Après, on discute deux problèmes rencontrés dans les centres de données cloud qui sont le vieillissement de logiciel et la gestion énergétique. La dernière partie de ce chapitre concerne les réseaux programmables où nous étudions l'architecture SDN/NFV.

— Dans le chapitre 3, nous allons présenter un état de l'art sur la modélisation stochastique pour l'analyse de la performance et de la sûreté de fonctionnement des systèmes. Les différents formalismes de modélisation et les méthodes analytiques afin d'obtenir les paramètres de performances seront présentés dans ce chapitre.

La deuxième partie concerne les principales contributions réalisées durant cette thèse. Cette partie est organisée en deux chapitres :

— Le chapitre 4 est consacré à notre première contribution concernant la modélisation et l'évaluation de la performabilité des systèmes de serveurs virtualisés (SVSs) et la gestion de l'énergie. La première partie introduit les différentes publications et études qui analysent le problème du vieillissement et le rajeunissement. La deuxième partie décrit notre modèle de SVS basé SRN, ainsi que les différentes hypothèses. La dernière partie, les résultats numériques seront présentés et discutés.

— Le chapitre 5 est destiné à notre deuxième contribution. Il porte sur les réseaux SDN/NFV et particulièrement le commutateur virtuel qui est un élément clé dans les centres de données virtualisés. Notre travail est une extension du travail de Baynat [132] où nous avons étudié le processus d'arrivée par groupes (batch arrival). La modélisation est basée sur un système de polling. Afin d'évaluer les performances du modèle considéré, nous avons opté pour une décomposition du système en un ensemble de files d'attente indépendantes avec des arrivées groupées et vacances du serveur.
Afin d'évaluer les performances du modèle, une évaluation basée sur le langage de programmation Matlab.

Le travail s'achève par une synthèse du travail mettant l'accent sur les perspectives à court et à long terme

1.4 Productions scientifiques

— M.Escheikh, Z. Tayachi, K. Barkaoui, "**Workload-Dependent Software Aging Impact on Performance and Energy Consumption in Server Virtualized Systems**", 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 111-118, 2016, doi :10.1109/ISSREW.2016.31.

— M.Escheikh; Z.Tayachi; K. Barkaoui (2018). "**Performability evaluation of server virtualized systems under bursty workload**", 14 th IFAC Workshop on Discrete Event Systems WODES 2018, IFAC-PapersOnLine. 51. 45-50. 10.1016/j.ifacol.2018.06.277.

— Z. Tayachi, M. Escheikh, K. Barkaoui. "**Performance evaluation of virtual switch with batch arrival using quasi-birth–death process**". 2019 International Conference on Industrial Engineering and Systems Management (IESM), Sep 2019, Shanghai, France. pp.1-6, 10.1109/IESM45758.2019.8948196⟩. hal-02475574⟩

— M.Escheikh ; Z.Tayachi ; K. Barkaoui, "**Performability modelling and analysis of server virtualised systems subject to workload-dependent software aging**", International Journal of Critical Computer-Based Systems (IJCCBS), Vol. 9, No. 3, 2019.

Chapitre 2

Contexte

Dans ce chapitre, nous discutons les différents domaines dans lesquels portent les contributions de cette thèse. Dans une première partie, nous présentons un aperçu de l'architecture et des services de cloud computing. La deuxième partie décrit la virtualisation, technologie permettant d'optimiser l'utilisation des ressources d'un centre de données. Après, nous discutons deux problèmes rencontrés dans le cloud computing qui sont la sûreté de fonctionnement et la gestion énergétique dans les centres de données cloud. Pour la sûreté de fonctionnement, nous traitons le problème du vieillissement de logiciel comme cause de panne et le rajeunissement comme une solution proactive. Afin de présenter le gaspillage de l'énergie, nous donnons quelques statistiques ainsi que les techniques de gestion dynamique. La dernière partie concerne les réseaux programmables où nous étudions l'architecture SDN/NFV.

Sommaire

2.1 Le Cloud Computing

2.1.1 Définition

Plusieurs définitions de Cloud Computing" ou "Informatique en nuage" ont été données dans la littérature [1], [2],[3],[4],[5]. Il n'existe pas une définition standard du Cloud Computing. Certaines sont focalisées sur la technique [6] alors que d'autres sont intéressées par l'aspect financier [1]. En se basant sur la définition donnée par le NIST (National Institute of Standards and Technology) [7] qui regroupe les différents concepts, le "Cloud Computing" est un modèle informatique représentant l'ensemble de ressources partagées, fournies à la demande sous forme de service et accessibles à travers internet de n'importe où dans le monde. Puisque les utilisateurs ne savent pas où ces ressources (réseau, serveurs, espace de stockage, applications ou services) sont situées ni comment elles sont fournies, on dit qu'elles se trouvent «dans le nuage». Le Cloud Computing rationalise l'utilisation des ressources configurables qui peuvent être rapidement allouées et dés-allouées en fonction des besoins, aussi il réduit les coûts d'investissement (le coût d'achat, d'installation et de maintenance des ressources).
Les utilisateurs ne paient que les services et les ressources utilisées (« payez uniquement ce que vous utilisez ») selon un accord de qualité de service (SLA pour Service Level Agreement) établi entre le client et le fournisseur de services Cloud. Dans ce contrat, le fournisseur de services Cloud s'engage d'assurer une qualité de service bien définie comme la disponibilité du service, le temps d'exécution, un délai maximal de réponse, etc.
Selon NIST, Le modèle de Cloud Computing est composé de :

— 5 caractéristiques principales : service à la demande, accès réseau universel, ressources mutualisées (les ressources sont partagées par plusieurs clients en suivant un modèle multiutilisateurs), élasticité rapide, service mesuré (une métrique utilisée et adaptée au type de service pour contrôler l'utilisation des ressources).

— 4 modèles de déploiement : privé (ressources utilisables uniquement par un client), communautaire (ressources partagées pour une communauté d'utilisateurs), public (ressources utilisées par tout les monde) et hybride (à la fois privé et public).

— niveaux de services : Infrastructure-as-a-Service (IaaS), Plateform-as-a-Service (PaaS) et Softwareas-a-Service (SaaS)

2.1.2 Les niveaux de services

Selon le type de service fourni, les services du Cloud peuvent être divisés en 3 modèles de services :

2.1.2.1 IaaS-Infrastructure-as-a-Service

C'est le service de plus bas niveau qui délivre des ressources d'infrastructure physiques ou virtuelles répondant aux besoins du client. Ces ressources sont gérées et hébergées dans des centres de données. Les ressources virtuelles sont des machines virtuelles (VM) si la technologie de virtualisation est basée sur un hyperviseur (par exemple, KVM, Xen) ou des conteneurs dans le cas de technologies basées sur des conteneurs (par exemple, des conteneurs Linux, LXC, Dockers). L'utilisateur final achète ces ressources comme un service et dispose du contrôle total de l'infrastructure allouée. Il gère, configure et exploite son système d'exploitation, toutes les couches supérieures, middlewares, bases de données, et ses applications. Les principaux fournisseurs de d'IaaS public sont Amazon Elastic Compute Cloud (EC2) [8] et Google Compute Engine (GCE) [9]. EC2 [10] met à disposition des machines virtuelles qui peuvent être déployées facilement et rapidement et un service de stockage « Amazon Simple Storage Service (S3) ». GCE est le composant IaaS de la plateforme Google Cloud Platform [11]. Les utilisateurs peuvent créer des machines virtuelles à la

demande. Les fournisseurs privés d'IaaS existent comme OpenStack [12], CloudStack [13], OpenNebula [14]. OpenStack [15] qui sont des plateformes libres et open-source. Cette plateforme est basée sur une architecture modulaire composée de plusieurs sous-projets, chacun est responsable d'un élément différent (gestion du réseau, du stockage, de la puissance de calcul, etc.). CloudStack [15] est un logiciel libre qui simplifie le déploiement par une architecture monolithique. Cette solution supporte les hyperviseurs les plus courants, comme KVM, XenServer, HYPER-V et VMware.

2.1.2.2 PaaS-Plateform-as-a-Service

Ce service, situé juste au-dessus du IaaS, fournit aux clients des plateformes de développement (un environnement de développement, des serveurs d'applications, une base de données, des API) permettant de développer et déployer leurs propres applications. Cette configuration est destinée principalement aux développeurs qui disposent d'une plate-forme de travail prête à l'emploi en évitant ainsi les tâches de gestion et de contrôle de l'infrastructure sous-jacente (serveur, réseau, stockage). Les fournisseurs de PaaS les plus populaires sont Google AppEngine [16] et Microsoft Azure [17].

2.1.2.3 SaaS-Software as a Service

Ceci est le plus haut niveau de services du Cloud. Le fournisseur de SaaS met à disposition des utilisateurs des applications prêtes à l'emploi et accessibles via une interface Web. L'utilisateur n'a pas besoin d'installer ces applications sur son ordinateur. L'administration, la configuration et le contrôle sont assurés par le fournisseur et non l'utilisateur. Ce service permet de réduire le coût d'achat, on paie juste ce qu'on utilise (ex. le temps d'utilisation de ces applications). Google est le fournisseur de SaaS le plus populaire avec ses applications solutions GoogleApps [18] (tels que Maps, Mail, Docs).

2.1.3 Modèles de déploiement

Selon la classification du NIST, il y a quatre modèles de déploiement des différents types de services du Cloud Computing.

- **Public :** l'infrastructure est ouverte au public et située en dehors des locaux des utilisateurs. Cette infrastructure appartient à une entreprise, une organisation universitaire ou gouvernementale ou une combinaison de ces entités. L'utilisateur évite ainsi les coûts d'investissement élevés mais il n'a pas un contrôle total sur ses données, le réseau et la sécurité.
- **Privé :** l'infrastructure est accessible par un seul utilisateur qui peut être une entreprise ou une organisation. L'utilisateur peut gérer l'infrastructure par lui-même, ou par des tiers. C'est le modèle le plus répandu. Il offre plusieurs avantages; l'utilisateur a un degré de contrôle plus élevé que celui offert par le Cloud public cependant, les coûts d'investissement sont plus élevés.
- **Communautaire :** l'infrastructure est partagée par un ensemble d'utilisateurs appartenant à une communauté avec des intérêts communs (politiques, objectifs, exigences de sécurité, etc.). Elle est gérée par une ou plusieurs organisations de la communauté, par une tierce partie ou par une combinaison de ces entités. Ce modèle de déploiement offre les avantages d'un Cloud privé comme la sécurité et la confidentialité. De plus, le principe de paiement est le même que Cloud public «payez ce que vous utilisez».
- **Hybride :** l'infrastructure est composée d'une combinaison d'infrastructures (privée, communautaire, public). Elles demeurent des entités distinctes mais interagissent par des technologies standard permettant une portabilité des applications et des données. Ce modèle de déploiement offre une grande flexibilité. En effet, l'utilisateur peut contrôler d'une manière fine l'infrastructure et la sécurité de ses données ainsi il fait des économies en profitant d'une tarification avantageuse.

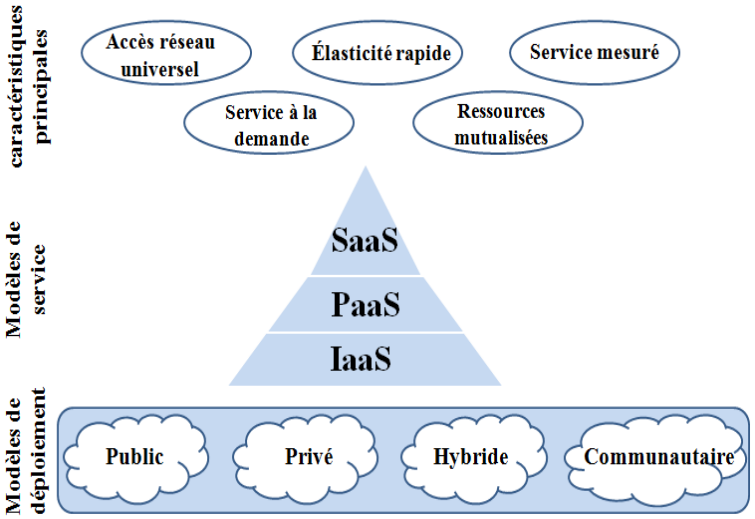


FIGURE 2.1 – Le cloud computing selon le NIST

2.1.4 Les systèmes IaaS

Au sein de l'IaaS, le client loue une infrastructure informatique localisée à distance auprès du fournisseur. Les ressources sont mises à la location sous forme de machines virtuelles, environnements d'exécution virtualisés. Selon un contrat établi entre le fournisseur d'IaaS et son client, le premier s'engage à s'occuper de l'infrastructure physique et garantir une disponibilité élevée et une meilleure qualité de service. Le second, le client, peut utiliser les machines virtuelles pour exécuter ses applications et paie ce qu'il utilise. Cela permet de réaliser des gains en termes de coût (le client n'a pas besoin d'acheter et de maintenir des matériels et le fournisseur transforme ses investissements en contrats de location) et de temps (pas de temps à passer à installer ou mettre à jour une machine).

2.1.4.1 Les caractéristiques

L'Infrastructure-as-a-Service dispose de plusieurs caractéristiques :

— accessibilité : le client peut accéder et gérer l'infrastructure facilement, de n'importe où (via internet), par des mécanismes standard.

— élasticité : Les ressources sont élastiques, pouvant être ajustées (augmenter ou réduire) dynamiquement et rapidement en s'adaptant aux besoins. Ces ressources peuvent être allouées ou libérées à tout moment et en toute quantité.

— disponibilité : les ressources IaaS sont accessibles à tout moment sans tenir compte de la possibilité d'une panne matérielle.

— optimisation : Afin d'optimiser l'utilisation des capacités matérielles, les ressources informatiques du fournisseur sont mises en commun entre plusieurs clients.

— gestion : le fournisseur contrôle les disponibilités des ressources matérielles alors que le client gère et contrôle les ressources qui lui sont allouées. Ils existent des techniques d'autogestion qui gèrent dynamiquement les ressources tels que DPM (Dynamic Power Management) ou DRS (Dynamic Resource Scheduling)

— interopérabilité : Un système de facturation est mis en place pour mesurer l'utilisation des ressources virtuelles et déterminer le montant à facturer pour chaque client.

TABLEAU 2.1 – Systèmes d'informatique en nuage

Fournisseur	modèles de services	modèles de déploiement
OpenStack	IaaS	Privé
OpenNebula	IaaS	Privé
CloudStack	IaaS	Privé
Amazon	IaaS(EC2)	Public
Microsoft Azure	IaaS(Virtual Machines) PaaS (Cloud Services)	Public
Google	IaaS(Compute Engine) PaaS (App Engine) SaaS (Apps)	Public

2.2 La virtualisation, la technologie du Cloud Computing

La virtualisation est la clé de voûte du Cloud Computing. Elle est présente dans tous les modèles de services. Le concept de la virtualisation est apparu dans les années 1960 [19]. Il a été développé par IBM afin d'utiliser efficacement les environnements matériels en les partitionnant en plusieurs environnements virtuels. La virtualisation, en informatique, est une technologie qui consiste à créer une version virtuelle d'un système d'exploitation, un périphérique de stockage, des ressources réseau, etc. Elle simplifie la gestion des ressources et des infrastructures en nuage, améliore l'utilisation des ressources en les partageant entre plusieurs utilisateurs et réduit les coûts en termes d'énergie et d'hébergement. Il existe plusieurs méthodes de virtualisation. Ces méthodes ont des objectifs différents et peuvent être classées en trois types.

2.2.1 Virtualisation des serveurs

La virtualisation des serveurs [20], [19] est une technologie qui permet d'exécuter en parallèle plusieurs systèmes différents sur un même serveur physique. C'est la consolidation de plusieurs serveurs virtuels appelés virtuelles machines (VM) en un seul serveur physique. La virtualisation des serveurs peut être basée sur un hyperviseur ou sur un conteneur. Si la virtualisation est réalisée via un hyperviseur (par exemple, Xen [21], VMware ESXi [22], [23], Microsoft Hyper-V [24], [25] et KVM [26].), chaque machine virtuelle (VM) aussi appelée instance, possède un système d'exploitation (OS) distinct (appelé OS invité) et ses propres ressources virtuelles telles que le processeur virtuel, la carte réseau virtuelle, la mémoire virtuelle et les disques virtuels [1]. Ces VMs sont indépendantes et isolées les unes des autres. De ce fait, si pour une raison particulière une VM devenait instable ou tombait en panne, cela n'impactera pas le bon fonctionnement des autres VMs. Elles sont gérés par l'hyperviseur appelé un moniteur de machines virtuelles (VMM) qui représente une couche d'abstraction en découplant la couche matérielle de la couche logicielle. La virtualisation basée sur un hyperviseur pourrait être classifié en trois catégories : virtualisation complète, paravirtualisation et virtualisation assistée par matériel. Ces catégories sont différées par la manière dont les OS hôtes et invités sont pris en charge et interagissent avec l'hyperviseur. Contrairement à la virtualisation basée sur l'hyperviseur, les solutions basées sur les conteneurs telles que Docker et LXC sont légères car elles sont au niveau applicatif. Chaque serveur physique exécute un noyau avec différents conteneurs isolés installés dessus.

2.2.1.1 Virtualisation des serveurs : Avantages inconvénients

L'avantage d'héberger plusieurs VMs dans un seul serveur est d'optimiser l'exploitation des ressources de ce dernier en mutualisant les services exécutés. De plus, la virtualisation dispose aussi de plusieurs avantages [27] :

— sécurité : Les machines virtuelles hébergées sur le même serveur sont isolées grâce au découplage entre applications et système d'exploitation et entre système d'exploitation et matériel.

— disponibilité : Les VMs peuvent également être migrées d'un serveur à un autre sans qu'il soit nécessaire de la stopper. Si un serveur tombe en panne, il est possible de déplacer les VMs automatiquement et rapidement.

— flexibilité : le déploiement de nouvelles instances et ressources de système est simple et rapide ce qui permet de faire face à une demande élastique.

— coût : la réduction des charges d'exploitation (OPEX) au travers la diminution du nombre de machines physiques mises en place. Ainsi réduire le nombre de machines physiques permet de réduire les coûts d'achat de matériel, les coûts de maintenance, l'encombrement (gain de place dans les centres de données) et indirectement les coûts d'énergie.

Néanmoins, la virtualisation des serveurs dispose des inconvénients et des défis à relever.

— complexité de l'administration : il est important de trouver des politiques de gestion efficaces pour s'adapter de manière agile à l'explosion de nombre de machines virtuelles.

— pannes matérielles : les machines virtuelles hébergées sur un serveur sont impactés par une défaillance de ce dernier.

— dégradations des performances : Une application s'exécutant à l'intérieur d'une machine virtuelle n'a pas les mêmes performances que celle qui est exécutée sur une machine non virtualisée (natif) [28] Aussi, ces dégradations sont causées par le problème du vieillissement logiciel.

2.2.1.2 Les différents types d'hyperviseurs

Un hyperviseur est une couche logicielle permettant d'héberger des VMs sur une même machine physique. Il gère le cycle de vie de la VM (création, configuration, exploitation et destruction). De plus, il gère l'allocation des ressources physiques aux VMs. On distingue deux types d'hyperviseur [29] (Fig.2.2).

— Les Hyperviseurs de type 1 : c'est un logiciel installé directement sur le matériel informatique. Il est dit un aussi hyperviseur "natifs" ou "bare metal". Ce type d'hyperviseur contrôle le matériel et le ou les systèmes d'exploitation invités. C'est la méthode de la virtualisation la plus performante qui est utilisée pour virtualiser l'infrastructure des centres de données Les hyperviseurs de type 1 les plus répandus sont VMware ESX/ESXi [30], Microsoft Hyper-V [31], XEN [32] et KVM [33].

— Les Hyperviseurs de type 2 : Contrairement aux hyperviseurs de type 1, les hyperviseurs de type 2 est un logiciel qui s'installe et s'exécute sur un système d'exploitation existant. C'est un hyperviseur simple à utiliser et à installer. Ainsi, l'installation d'une VM est rapide et facile. En revanche, cet hyperviseur est moins performant que celui de type 1 car il y a moins de ressources disponibles pour les VMs. Les hyperviseurs de type 2 les plus connus sont VMware Workstation/Player [34], Microsoft Virtual PC [35] et Oracle VirtualBox [36].

2.2.2 Virtualisation du stockage

Le concept de virtualisation du stockage est similaire à celui de la virtualisation de serveurs. Il consiste à regrouper la capacité de stockage de plusieurs appareils de stockage en réseau sous forme d'un seul appareil de stockage (virtuel) administré depuis une console centrale. Cela offre une grande flexibilité pour augmenter la capacité de stockage en fonction des besoins et une facilité de migration des données. Ainsi l'utilisateur a un vue globale sur l'intégralité du stockage permettant une gestion simple des données.

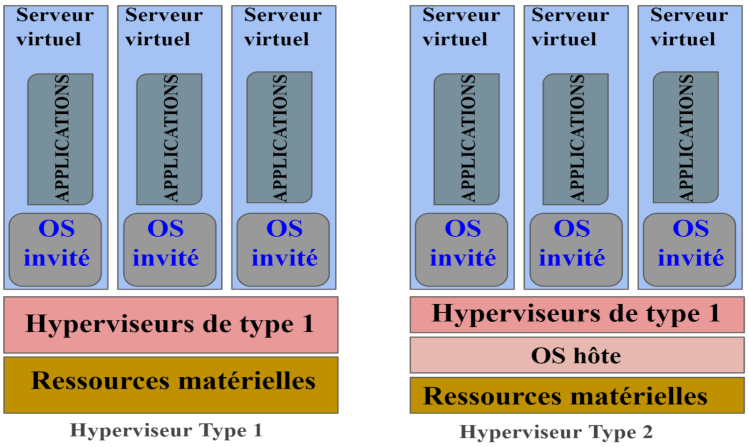


FIGURE 2.2 – Les différents types d'hyperviseurs

2.2.3 Virtualisation de réseau

La virtualisation de réseau permet d'exécuter des réseaux logiques sur un réseau physique partagé. De la même manière que la virtualisation de serveurs qui virtualise les ressources (vCPU et vRAM), la virtualisation de réseau virtualise les cartes d'interface de réseau (NIC), des commutateurs et routeurs ainsi des fonctionnalités réseau, telles que des équilibreurs de charge et des pare-feu. Dans les environnements du Cloud, la virtualisation du réseau est souvent associée à la virtualisation des ressources pour fournir aux utilisateurs des plates-formes virtualisées. De nombreuses technologies ont été développées pour fournir la virtualisation de réseau. Parmi les technologies les plus courantes, nous citons Vlan, VPN, VXlan,...

2.3 Sûreté de fonctionnement et Tolérances aux pannes

La complexité croissante des systèmes, des protocoles, des architectures des réseaux de communication et des services, et les exigences sous-jacentes en termes de performance, de sûreté de fonctionnement et de résilience ont davantage catalysé l'intérêt porté aujourd'hui à leur modélisation et à leur vérification. L'analyse de performance et la sûreté de fonctionnement représentent aujourd'hui des disciplines scientifiques en plein essor. C'est un concept intégrateur qui englobe un ensemble d'attributs qui permettent d'exprimer les propriétés attendues du système, évaluer les risques potentiels, et tenter à minimiser leurs conséquences lorsqu'elles se présentent. Ces attributs sont :

— La fiabilité, c'est la capacité du système d'assurer la continuité de service correct pendant une durée donnée dans des conditions données.

— la disponibilité, représente la probabilité que le système soit opérationnel à un instant donné

— la sécurité, C'est l'absence d'accidents catastrophiques sur l'utilisateur ou son environnement

— L'intégrité concerne la non existence d'une mauvaise modification du système.

— La maintenabilité c'est la souplesse du système vis-à- vis les modifications et des réparations qu'il peut subir.

2.3.1 Tolérances aux pannes

Malgré que les multiples technologies utilisées dans le cloud computing afin d'assurer un service continu aux utilisateurs, ce paradigme est toujours vulnérable à un grand nombre de pannes

système. Ainsi, les fournisseurs de service sont de plus en plus préoccupés par la fiabilité et la disponibilité des services. La tolérance aux pannes et la résilience sont des moyens efficaces pour répondre à ces problèmes.
Dans les centres de données, la technologie de virtualisation est utilisée pour provisionner des ressources informatiques (par exemple, en fournissant des machines virtuelles avec une quantité donnée de CPU, de mémoire et de capacité de stockage).// Par conséquent, un hôte physique est souvent utilisé comme un ensemble de plusieurs hôtes virtuels par le fournisseur de services. Cependant, l'utilisation de composants de base expose le matériel à des conditions pour lesquelles il n'a pas été conçu à l'origine. De plus, en raison de la nature extrêmement complexe de l'infrastructure sous-jacente, même les centres de données soigneusement conçus sont sujets à un grand nombre de pannes. Ces défaillances réduisent évidemment la fiabilité et la disponibilité globales du service de cloud computing. En conséquence, la tolérance aux pannes est d'une importance primordiale pour les utilisateurs ainsi que pour les prestataires de services pour assurer un fonctionnement correct et continu du système même en présence d'un nombre inconnu et imprévisible de pannes. En général, une panne ou une faille (un échec) représente la condition dans laquelle le système ne remplit pas sa fonctionnalité prévue ou le comportement attendu. Un échec se produit en raison d'une ou plusieurs erreurs.
Dans l'environnement de Cloud Computing, une défaillance dans une couche donnée a normalement un impact sur les services offerts par les couches supérieures. Par exemple, une défaillance dans un middleware au niveau de l'utilisateur (PaaS) peut produire des erreurs dans les services logiciels construits dessus (applications SaaS). De même, les pannes du matériel physique ou de la couche IaaS auront un impact sur la plupart des services PaaS et SaaS. Cela implique que l'impact des pannes dans la couche IaaS ou le matériel physique est significativement élevé. Pour remédier à ces problèmes,il est important de développer des solutions qui permettent de remplir les fonctions du système et d'éviter la faille totale du système même en présence de pannes. Des techniques de tolérance aux pannes ont été mises en œuvre.
La tolérance aux pannes est définie comme la capacité du système à tolérer les défauts de façon à ne pas engendrer des interruptions de service. Cela implique qu'il est de la plus haute importance de comprendre clairement et de définir ce qui constitue le comportement correct du système afin de définir les caractéristiques de défaillance et ainsi un système tolérant aux pannes sera développé. Des multiples techniques ont été conçues comme la prévention des pannes, le masquage des pannes et le recouvrement après panne, afin de prévenir les défauts et d'éviter l'interruption du système. La préventions des pannes utilisent des méthodes d'implémentation qui empêchent les erreurs introduites lors de la phase du développement. On peut citer la technique de migration comme une technique de prévention des pannes.
Le masquage des pannes est une technique proactive qui a comme objectif de réduire la probabilité que le système tombe en panne en dupliquant les services et serveurs sur différents dispositifs. Ainsi la panne est masquée par la duplication en cas de faille du système.
La méthode du recouvrement après panne consiste à stocker l'état du système périodiquement sur un support de stockage stable et fiable. Une fois le système tombe en panne, celui-ci sera redémarré à partir du point de sauvegarde.

2.3.2 La tolérances aux pannes dans le Cloud

Beaucoup de travaux ont été réalisés pour traiter la tolérance aux pannes dans l'environnement Cloud Computing. La virtualisation joue un rôle primordial dans la gestion des défaillances. Des techniques de gestion des VMs ont été proposées comme la mise en pause, la reprise et la migration des VMs d'un hôte source vers un autre plus robuste. En effet, lorsqu'on détecte une panne, un mécanisme de migration sera lancé tout en gardant les images des VMs. Bien que la migration des VMs présente plusieurs avantages permettant une continuité de service, elle a aussi des inconvénients. Un surcoût important peut être engendré par la migration des VMs, particulièrement, si l'image entière de VM doit être émigrée. Aussi, des techniques et des algorithmes

complexes doivent être mis en place afin de prévenir les pannes et éviter la migration au moment de la défaillance du nœud sur lequel elle s'exécute.

2.4 Vieillissement et rajeunissement logiciel dans les SVSs

Le vieillissement des logiciels représente l'une des causes des pannes les plus répandues dans le cloud computing. C'est un phénomène qui affecte de nombreux systèmes logiciels complexes de longue durée, qui présentent une dégradation des performances ou un taux d'échec croissant. Plusieurs stratégies basées sur le rajeunissement proactif de l'état du logiciel ont été proposées pour contrer le vieillissement du logiciel et éviter les pannes.

2.4.1 Vieillissement logiciel

Les études montrent que la défaillance des systèmes informatiques est due aux défaillances logicielles qu'aux défaillances matérielles. Au fur et à mesure que le temps d'exécution s'écoule, un système logiciel présente des erreurs telles que la dégradation du logiciel, le blocage, la fuite de mémoire, la fragmentation de la mémoire, une corruption des données, une fragmentation de l'espace de stockage et une série d'erreurs. Ceci constitue un phénomène appelé le vieillissement du logiciel [37] [38] [39]. Le vieillissement du logiciel signifie que le logiciel a vieilli et qu'il montre des signes de vieillissement, comme un crash ou un blocage. Il présente également une fuite de mémoire, une dégradation du niveau de performance et l'augmentation du taux de défaillance.
Le phénomène de vieillissement des logiciels a été observé depuis les années 1960, il a été observé dans de nombreux systèmes et des applications critiques pour l'entreprise et la sécurité. Elle est causée par ce que l'on appelle des bugs liés au vieillissement, un type de bugs qui entraîne l'accumulation des erreurs et une consommation progressive de ressources, comme la mémoire physique. Cela entraîne des pannes qui peuvent causer d'énormes pertes économiques ou des risques pour la vie humaine [40]. Le système militaire Safeguard était affecté par des blocages survenant une fois que les tampons de rapport d'erreur étaient pleins [41]. Au fur et à mesure que les logiciels gagnaient en taille et en complexité, le vieillissement des logiciels a été observé dans un nombre croissant de systèmes de longue durée, y compris les logiciels de commutation de télécommunications et de facturation ([42] ; [42]), et cela a conduit à l'accident bien connu du système antimissile Patriot qui a causé la perte de vies humaines [43].
Le vieillissement des logiciels peut être attribué à des bogues logiciels insaisissables : des études au début des années 1990 de Bernestein sur les systèmes de télécommunications [44] ont souligné la forte incidence de bogues qui, lorsqu'ils sont déclenchés, ne provoquent pas immédiatement une défaillance logicielle, mais se manifestent sous forme de fuite de mémoire, verrous de fichiers, corruption de données et accumulation d'erreurs numériques, ce qui fait que le système dégrade lentement ses performances et finit par échouer. Souvent, ces bogues sont trop subtils ou trop coûteux pour être supprimés au cours du développement.
Le vieillissement des logiciels se manifeste aussi à différents domaines, tels que le système d'exploitation ([45],[46]), les systèmes temps réel, les serveurs Web [47], les systèmes informatiques en nuage , la machine virtuelle et le moniteur de machine virtuelle [48], [49].

2.4.2 Rajeunissement logiciel

Comme il est difficile d'identifier et d'éliminer les causes du vieillissement de logiciel, la recherche aux laboratoires AT&T Bell sur les logiciels tolérants aux pannes a ensuite identifié le rajeunissement des logiciels comme une solution rentable pour contrer le vieillissement des logiciels [50] ; [51] ; [50].
Le rajeunissement logiciel est une approche proactive pour prévenir la dégradation des performances et les pannes dues au vieillissement logiciel : il consiste en un nettoyage occasionnel ou

périodique des effets du vieillissement (qui peut être obtenu par un simple redémarrage du logiciel, ou par des techniques plus complexes), dans l'ordre pour reporter les échecs et restaurer les performances. Elle nettoie et restaure l'état de l'environnement de l'application en supprimant les erreurs accumulées, libérant les ressources du système et améliorant la fiabilité des logiciels [52]. Ce type de gestion proactive de pannes est principalement utilisé pour la haute disponibilité du logiciel, mais aussi pour des systèmes critiques exigeants en termes de sécurité.

2.4.2.1 Politiques de rajeunissement

On distingue deux types de politiques de rajeunissement :

— politique périodique (à base de temps) : le rajeunissement sera déclenché après chaque intervalle de temps déterministe ;

— politique prédictive/proactive : le déclenchement du rajeunissement est estimé sur la base de la collecte et l'analyse statistique des données du système. Dans ce cas, les métriques du système sont surveillées en permanence.

— Politique composite/hybride : C'est la combinaison de deux politiques précédentes.

2.4.3 Vieillissement et rajeunissement dans le cloud

La virtualisation des serveurs est considérée comme une technologie clé sur laquelle repose le Cloud Computing en fournissant des infrastructures logicielles. Elle permet d'instancier une ou plusieurs VMs sur une seule machine physique gérée par des VMM. En raison de la complexité croissante des systèmes liés au cloud, le vieillissement logiciel peut être observé lors du déploiement du logiciel. En outre, le VMM et les VMs comme tout autre logiciel, entraînent des problèmes de vieillissement du logiciel long de leur exécution. Ceci peut avoir des conséquences néfastes sur la disponibilité des systèmes virtualisés. Ainsi, les chercheurs de la communauté SAR (Software Aging Rejuvenation) ont commencé à étudier le problème du vieillissement de ce type de systèmes. De nombreuses études ont été menées sur le vieillissement dans le cloud et, plus largement, sur les systèmes virtualisés pour atténuer les effets du vieillissement des logiciels dans VM et VMM, prévenir l'échec le plus probable et surtout pour garantir de hauts niveaux de continuité des services. Afin de réduire le temps d'arrêt de VM, plusieurs techniques et politiques ont été développées pour un rajeunissement logiciel en minimisant les coûts du rajeunissement en termes de temps d'arrêt ou de transactions perdues.

2.4.4 Corrélation entre rajeunissement VMM et rajeunissement VM

Comme le VMM gère les VM, il est évident que l'exécution de VM dépend étroitement de celui du VMM. Ainsi, le rajeunissement VMM peut entraîner des coûts supplémentaires dans la mesure où il affecte la disponibilité des VMs. En effet, quand un VMM est rajeuni, l'exécution des VMs hébergées sur le VMM peut également être interrompue [53], [54]. Les techniques de rajeunissement VMM se déclinent en trois approches différentes :

— **Rajeunissement Cold-VM :** c'est l'approche la plus simple et la plus utilisée qui consiste simplement à arrêter toutes les VMs hébergées avant déclencher le rajeunissement du VMM. Il redémarre les VMs une fois le rajeunissement est terminé. Cette technique impose un arrêt des VMs et peut faire perdre les transactions courantes, s'exécutant sur ces VMs. Mais aussi ces dernières seront rajeunies indirectement en effaçant l'ensemble des états de vieillissement.

— **Rajeunissement Warm-VM :** cette approche consiste à stocker l'état d'exécution des VM dans la mémoire persistante et le reprendre après le rajeunissement du VMM. Comparée au Cold-VM, la technique de rajeunissement Warm-VM peut être effectuée rapidement par un mécanisme de suspension / reprise en mémoire (l'image en mémoire des VM est conservée

en RAM lors du redémarrage du VMM plutôt que sur le stockage [55]). Les auteurs dans [55]et [56] ont montré aussi que le rajeunissement Warm-VM améliore la disponibilité de l'application hébergée sur les VM.

— **Rajeunissement Migrate-VM :** dans cette technique, le temps d'arrêt est encore réduit par rapport aux deux techniques précédentes, en migrant une VM en cours d'exécution sur un hôte vers un autre hôte pendant le rajeunissement de VMM, ce qui rend la VM disponible. C'est la migration en direct. Cette approche ne rajeunit pas les VM et elle est limitée par la capacité des autres hôtes à accepter les VM migrés. Il existe plusieurs variantes de la technique de rajeunissement Migrate-VM qui peuvent être divisées en fonction du type de migration en direct de VM (arrêt-et-copie (stop-and-copy) ou pré-copie (pre-copy)) et des stratégies de retour à l'hôte d'origine (retour (return-back) ou rester (stay-on)). La migration stop-and-copy arrête les opérations de la VM et copie le contenu de la mémoire sur le serveur de destination. Dans la variante de pre-copy, la mémoire de la VM est copiée sur le serveur de destination sans arrêt de son fonctionnement. Après l'achèvement du rajeunissement du VMM, la VM peut retourner à l'hôte d'origine en suivant la stratégie return-back. Cette dernière consiste à migrer les VM vers l'hôte d'origine. Aussi, la VM peut suivre la stratégie de stay-on qui permet à la VM migrée de continuer son exécution sur le nouveau serveur.

2.5 Gestion énergétique dans le cloud computing

Comme on l'a vu dans les sections précédentes, le cloud computing représente une solution pour répondre aux besoins des utilisateurs et des entreprises grâce à sa puissance de calcul illimitée, sa disponibilité, ses tarifs avantageux, ses services à la demande et sa qualité de service. Ces services sont hébergés au sein des centres de données Cloud(Cloud Datacenters). Un centre de données désigne un bâtiment destiné à héberger de façon sécurisée des centaines de milliers de serveurs fonctionnant en continu. Il est composé d'un réseau d'ordinateurs, d'espaces de stockage et les équipements d'interconnexion. Afin de garantir un fonctionnement continu, une alimentation électrique continue est nécessaire pour le fonctionnement et le refroidissement des appareils. Par conséquent, les centres de données sont considérés comme des stations gourmandes en énergie puisqu'ils consomment d'énormes quantités d'énergie électrique. Cette énorme consommation entraîne des coûts d'exploitation élevés ainsi l'émission d'une quantité importante de dioxyde de carbone (CO2) dans l'environnement. A titre indicatif en 2015, la consommation d'électricité nécessaire au fonctionnement des 182 data centers français était supérieure à celle de la ville de Lyon. Selon le centre Observation, impacts et énergie (OIE) de l'école des Mines ParisTech, cette consommation a été multipliée par 4,5 entre 2011 et 2016. D'ici 2030, l'alimentation électrique des centres de données représentera 3% de l'électricité mondiale. En France, une estimation montre que les centres de données consomment entre 7% et 10% de la production électrique nationale. Pour ces raisons, l'efficacité énergétique est devenue l'un des plus gros problèmes rencontrés par les centres de données.
Il est alors important de mettre en place des mécanismes permettant de gérer efficacement ce problème. Pour une bonne maîtrise de l'énergie dans les centres des données, les études se sont réalisées à ses différents composants tels que les serveurs et le système de refroidissement. Au départ, les chercheurs se concentraient sur les systèmes de refroidissement. Des approches ont été mises en place ; consistent à choisir l'emplacement approprié du centre des données de telle sorte qu'on réduit la température en utilisant de l'air ou de l'eau naturellement fraîche au lieu de la réfrigération mécanique. Dans certaines conditions climatiques, un gain en énergie majeur peut être obtenu. D'autres études ont été orientées vers la consommations d'énergie des serveurs puisque plusieurs recherches ont montré qu'il existe un gaspillage d'énergie côté serveur. Ceci peut être réalisé en améliorant à la fois l'infrastructure physique des centres de données et les algorithmes d'allocations et de gestion des ressources.

2.5.1 Problème de gaspillage d'énergie et des ressources

La plupart du temps les serveurs fonctionnent à 10-50% de leur pleine capacité, conduisant à des dépenses supplémentaires relatives au sur-dimensionnement (overprovisioning) et donc un coût total d'acquisition (TCA) supplémentaire. En outre, la gestion et le maintien des ressources sur-dimensionnées conduit à une augmentation du coût total de possession (TCO). D'autre part, le problème de la faible utilisation des serveurs est aggravé par des gammes étroites de puissance dynamiques de serveurs. En effet les serveurs, même complètement inactifs, continuent à consommer jusqu'à 70% de leur puissance de crête. Par conséquent, le fait de garder les serveurs sous-utilisés implique une très grande inefficacité énergétique.
Dans une étude statistique sur l'un de ses centres de données hébergeant 5000 serveurs, Google constate que les capacités CPU de la majorité des serveurs ne sont exploitées qu'entre 10% et 50% [57]. Cette sous utilisation des ressources entraîne un gaspillage énergétique très important tant au niveau des serveurs physiques qu'au niveau du centre de données. Ceci s'explique potentiellement par une consommation électrique des serveurs qui évolue de manière non proportionnelle à son activité. En effet un CPU dans un état idle consomme entre 50% à 80% de sa consommation électrique en pleine charge [58]. Ce comportement engendre une consommation électrique sans valeur ajoutée qui se répercute lourdement sur le bilan énergétique de l'ensemble du centre de données. Une autre étude menée par IBM sur un centre de données non virtualisé composé de 100 unités énergétiques [59], montre que 55% de ces unités sont effectivement utilisées pour assurer le bon fonctionnement des serveurs (générateurs, climatisation) alors que 45% de ces unités sont exploitées par les serveurs. Cette étude a montré aussi que 30% de la consommation énergétique des serveurs est utilisée par le CPU et les 70% restantes sont accaparées par des composants autres que le CPU (ventilateur, convertisseurs électriques, périphériques de stockage). Le CPU ne consomme donc que 13.5% de l'énergie globale. Etant donnée que le serveur est le plus souvent non chargé, donc en mode idle, uniquement 3% de la quantité d'énergie utilisée par un centre de données est exploitée pour faire tourner des applications et les 97% restantes sont consommées pour assurer le refroidissement des serveurs.

2.5.2 Techniques de gestion dynamique de l'énergie

Le problème de la gestion des ressources énergétiques a été examiné dans la littérature pour différents contextes, au niveau composant, au niveau OS de serveurs virtualisés et non virtualisés, mais aussi au niveau cluster, data centers et cloud.
Une façon idéale d'aborder le problème de l'inefficacité énergétique est d'adapter l'énergie livrée proportionnellement à la charge de travail de l'application. On distingue trois approches de gestion de l'énergie :

2.5.2.1 Approche DVFS

Cette approche se base sur l'activation du passage à l'échelle dynamique de la tension et de la fréquence DVFS (Dynamic Voltage Frequency Scaling), même si les applications sont actives [60]. Le principe de cette technique consiste à limiter les performances du processeur au moment où il consomme trop d'énergie alors que la tâche peut être accomplie en traitant à une fréquence plus basse. La tension et la fréquence ont une relation directe. Notons que la consommation de l'énergie des processeurs est proportionnelle à la fréquence de l'horloge et au carré du voltage qui leurs sont fournis [61].
Il a été observé que le serveur consomme environ 70% d'énergie lorsqu'il est inactif, économiser cette énergie aura un impact important sur la réduction d'énergie. DVFS peut gérer ce problème efficacement. Il permet l'ajustement dynamique de la tension et de la fréquence du CPU sur la base de la demande (charge de travail) courante des ressources. La réduction de la consommation d'énergie peut ainsi être réalisée par la commutation des serveurs inactifs à des modes de faibles

puissances (sommeil,..), éliminant ainsi la consommation d'énergie au repos (la puissance statique) et réduisant la consommation globale de l'énergie du système.
DVFS est essentiellement conçu pour l'efficacité énergétique dans un système embarqué. Mais cela peut être mis en œuvre et considérer l'une des techniques d'efficacité énergétique les plus efficaces pour un serveur de calcul intensif.

2.5.2.2 La gestion des ressources

Cette technique consiste à sélectionner les ressources informatiques telles que le calcul, le stockage, le réseau de manière intelligente. Ces ressources sont allouées à la demande en attribuant aux utilisateurs les meilleures ressources possibles en matière de performances et de prix en regardant le SLA. Lorsqu'une demande est reçue, un ordonnanceur identifie une allocation de ressources optimale en regardant et en analysant l'état actuel du système [62]. L'ordonnancement s'intéresse principalement à la gestion des machines virtuelles plus précisément aux migrations des VM. Le choix des VMs se base sur trois critères : performances, équilibrage de charge et efficacité énergétique.
La conservation de l'énergie peut être réalisée, en implémentant au niveau de l'hyperviseur, des algorithmes optimisant le placement et la consolidation.

Approche algorithmique basée sur la consolidation Le placement des VMs est essentiel pour la conservation de l'énergie, dans la mesure où la tâche affectée à la VM a des exigences en termes de ressources telles que la mémoire, le processeur, le réseau et le stockage, et l'énergie utilisée est directement proportionnelle à la consommation des ressources. Les algorithmes de placement ont pour rôle d'allouer efficacement les VMs aux serveurs. Une fois que les VMs ont terminé d'exécuter leurs tâches, les ressources du serveur peuvent être réallouées aux VMs. Si le nombre de VMs requis et la charge sur les serveurs dècroient, les VMs restantes peuvent être consolidées sur un nombre minimum de serveurs au travers la migration, et les serveurs inutilisés peuvent être mis dans des états de faibles consommation d'énergie, ou carrément éteints.
La consolidation dynamique des VMs se compose de deux processus fondamentaux : faire migrer des VMs à partir d'hôtes sous-utilisés pour réduire au minimum le nombre d'hôtes actifs ; et décharger les VMs à partir d'hôtes lorsque ceux-ci sont surchargés, pour éviter la dégradation de la performance subie par les VMs.

2.5.2.3 Consolidation de la charge de travail

Les centres de données sont physiquement répartis sur une zone géographique. Lorsqu'une demande est reçue, elle est placée dans le centre de données situé géographiquement le plus proche. Cela aide à réduire les retards du réseau, ce qui à son tour réduit les délais d'exécution. À l'intérieur du centre de données, on essaie de choisir le bon nœud pour le traitement de la demande en utilisant le PM actuellement actif pour économiser de l'énergie. Les études montrent que 70% du temps les serveurs sont sous-chargés (en état idle) et cela entraîne une consommation de plus de 50% de l'énergie. Afin de traiter le problème de gaspillage de l'énergie, une stratégie est mise en place qui met le nœud sous-chargé dans un état de basse énergie ou à l'arrêt. De même, si une nouvelle demande arrive et qu'aucun serveur actif ne peut s'exécuter, un nouveau serveur doit être allumé ou réveillé pour répondre à cette demande.

2.6 Les réseaux programmables

Cette section présente les deux concepts clés, les réseaux définis par logiciel (Software-Defined Networking(SDN)) et la virtualisation des fonctions réseau (NFV), qui redéfinissent les réseaux. Ces deux paradigmes introduisent des nouvelles propriétés et des fonctionnalités aux architectures traditionnelles de réseau comme la programmabilité et la virtualisation des réseaux. SDN

et NFV offrant une grande flexibilité permettant de rationaliser l'utilisation des ressources disponibles.

2.6.1 Les réseaux définis par logiciels (SDN)

Le réseau défini par logiciel est un paradigme émergent qui promet de surmonter les difficultés d'implémentation, de gestion et de modification de l'infrastructure réseau en découplant les fonctions de contrôle (plan de contrôle) et de transfert des données du réseau (plan de données). SDN apporte plusieurs avantages. Il introduit une séparation entre le plan de contrôle et les routeurs et commutateurs sous-jacents. Cela offre une flexibilité et une programmabilité des réseaux afin de rendre sa gestion simple et d'améliorer les performances du réseau. Ainsi, les périphériques réseau se transforment en simples périphériques de transfert gérés par le plan de contrôle [63].

2.6.1.1 Architecture SDN

L'Open Networking Foundation (ONF) a donné la définition suivante [64] :
"In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications."
L'architecture SDN découple les fonctions de contrôle (control planes) et de transfert des données du réseau (data plane) et permet intelligence du réseau logiquement centralisés afin d'avoir une infrastructure complètement exempte de tout service réseau.
En se basant sur cette définition, l'architecture SDN est composée de trois plans principaux.

- **Plan de données :** C'est le niveau le plus bas, appelé également plan d'infrastructure. Il comprend des périphériques de transfert, notamment des commutateurs matériels et logiciels, des routeurs et des pare-feu. Ce plan est responsable de la transmission du trafic selon un ensemble de règles [65].
- **Plan de contrôle :** c'est le niveau central et comprend tous les contrôleurs considérés comme le cœur du réseau. Le contrôleur est une entité logique centralisée qui contrôle et surveille l'état du réseau. Il donne une vue unique et globale du réseau, ce qui simplifie l'administration et la configuration de périphériques de bas niveau et la détermination du comportement du trafic réseau. Le plan de contrôle orchestre la demande des applications en injectant des règles de transmission spécifiques dans les périphériques de transfert. La connexion entre le plan de contrôle et le plan de données a été établie via une interface de programmation d'application (API) appelée southbound API (SBI)qui est une interface descendante permettant la communication avec le niveau inférieur (plan de données). La SBI la plus populaire est OpenFlow [66], mais ce n'est pas le seul.
- **Plan d'application :** il s'agit du niveau le plus élevé de l'architecture SDN. Il comprend différentes applications telles que les applications professionnelles, la surveillance de la sécurité et le contrôle d'accès. Ces applications permettent une gestion souple et agile du réseau.

Plusieurs travaux ont traité le déploiement du SDN sur différents réseaux, tels que le cloud computing [67], Edge Computing ou traitement des données à la périphérie du réseau [68], l'Internet des objets (IoT) [69], [70], les systèmes cyberphysiques [71] et villes intelligentes [72], réseaux de transport [26], réseaux sans fil [27]. Chaque travail a traité des aspects différents, tels que l'architecture, la mise à jour du réseau SDN, la sécurité, la fiabilité et l'équilibrage de la charge.

2.6.2 La virtualisation des fonctions réseau (NFV)

L'évolution des demandes des clients nécessitent de nouveaux services. Cependant, dans les réseaux traditionnels, l'hétérogénéité des équipements et l'incompatibilité entre les différentes

plates-formes et les technologies de fabrication ont causé une complexité de créer et de déployer de nouveaux services et une augmentation des coûts CAPEX et OPEX pour les fournisseurs de services et d'infrastructures [73]. Pour surmonter ces difficultés, une technologie appelée «virtualisation des fonctions réseau (Network Function Virtualization (NFV))» a été proposée par l'Institut européen des normes de télécommunications (ETSI). Cette solution exploite les avantages de la virtualisation en permettant aux services de réseau qui sont contenus dans les routeurs, pare-feu, équilibreurs de charge et autres, à être hébergés sur des (VM). Cela offre une grande flexibilité dans la création et la gestion des services réseau. Ainsi, NFV réduit les coûts CAPEX et OPEX car les administrateurs réseau n'auront plus besoin d'acheter du matériel dédié.
L'architecture NFV définie par l'ETSI comprend trois composantes fonctionnelles principales [74] :

— **Fonction de réseau virtualisée** (Virtualized Network Function VNF) : correspond aux fonctions réseaux virtualisées pouvant être déployées sur une infrastructure de virtualisation de fonction de réseau (NFVI) et pouvant fonctionner indépendamment des autres.

— **Insfrastructure NFV :** (Network Function Virtualisation Infrastructure NFVI) comprend toutes les ressources matérielles (serveurs, cartes électroniques, ...) et le logicielles. NFVI inclut la connectivité de calcul, de stockage et réseau virtualisée.

— **Gestion et orchestration NFV**(M&O (Management and Orchestration)) est responsable de la gestion et du contrôle des ressources matérielles et / ou logicielles, de l'orchestration des services réseau et de la gestion du cycle de vie VNF (par exemple, initialisation, suspension et terminaison). Cette couche agit comme un plan de contrôle.

2.6.2.1 SDN/NFV

Initialement, les deux paradigmes de réseau SDN et NFV ont été développés indépendamment. Cependant, ils partagent des objectifs communs et des techniques similaires [75]. SDN et NFV représentent bien des technologies complémentaires et une forte synergie existe entre elles. Chacun peut tirer parti de l'autre, ce qui permet une flexibilité et une simplicité des réseaux. D'une part, SDN simplifie la gestion du réseau et accélère le déploiement de NFV grâce à une configuration flexible et automatisée des nœuds VNF via un contrôleur centralisé [76]. D'autre part, NFV peut implémenter un contrôleur SDN s'exécutant sur une machine virtuelle. Cela signifie que les contrôleurs SDN peuvent être exécutés en tant que fichiers VNF et ainsi bénéficier des fonctionnalités NFV telles que la fiabilité et l'évolutivité. Plusieurs travaux de recherche ont intégré NFV et SDN dans différents environnements (Cloud Computing, réseau étendu WAN, 5G, etc.). Cependant, la diversité des composants de réseau et leur interaction complexe constituent un défi pour les architectures SDN/NFV. Pour ces raisons, plusieurs études de recherche industrielles et universitaires attaquent les problèmes de fiabilité et les performances de l'architecture SDN/NFV [77], [78].

2.7 Conclusion

Ce chapitre a présenté le concept Cloud Computing, qui a révolutionné la façon dont les services informatiques sont mis à disposition et gérés. C'est un nouveau paradigme qui offre des ressources informatiques à la demande grâce à la technologie de la virtualisation. Mais avec la complexité des systèmes et des protocoles, la sûreté de fonctionnement présente un défi pour les fournisseurs de Cloud. Nous sommes intéressés par le vieillissement des logiciels qui représente l'une des causes des pannes dans le Cloud computing qui peut provoquer des résultats catastrophiques et comme solution proactive le rajeunissement logiciel.
Comme nous savons que les services Cloud sont hébergés au sein des centres de données Cloud (Cloud Datacenters) de façon sécurisée. Cependant, ces centres consomment une grande quantité d'énergie afin de garantir un fonctionnement continu. Nous avons présenté le problème de gaspillage d'énergie et des ressources et nous avons introduit les différentes techniques de gestion

dynamique de l'énergie. Ces techniques seront utilisées pour répondre à nos contributions.
Dans la dernière partie de ce chapitre, nous avons représenté le nouveau paradigme d'architecture réseau (SDN : Software Defined Networks) reposant sur la virtualisation des ressources et des fonctions réseaux qui permet de surmonter les difficultés d'implémentation, de gestion et de modification de l'infrastructure réseau.

Chapitre 3

État de l'art sur la modélisation stochastique pour l'analyse de la performance des systèmes

L'étude du comportement d'un système réel dans un environnement opérationnel est difficilement réalisable en raison de difficultés pratiques et de coûts. Pour ces raisons, la modélisation des systèmes est considérée comme une étape fondamentale dans la conception et le développement des systèmes. La résolution des modèles permet la prédiction et l'évaluation des performances des systèmes.
Dans cette thèse, notre objectif est de renforcer des axes de recherche se rapportant à la modélisation pour l'analyse des performances des systèmes virtualisés Cloud. Ce chapitre présente une synthèse des différents formalismes de modélisation et les méthodes analytiques afin d'obtenir les paramètres de performances.

Sommaire

3.1 La modélisation

3.1.1 Processus de la modélisation

La modélisation d'un système réel consiste à le décrire en utilisant des méthodes mathématiques. Elle permet de représenter le fonctionnement d'un système par des modèles dans le but de mieux appréhender et cerner son comportement et sa dynamique. La représentation d'un système est souvent réalisée en utilisant des modèles et des outils appropriés permettant d'approcher son comportement. Chaque outil représente une approche de modélisation analytique. Le choix de l'approche, et par la suite de l'outil, dépendent de la nature du système à étudier et du type de performance à analyser.
La modélisation est basée sur des hypothèses spécifiques permettant de fournir une vue abstraite des systèmes exploitables pour différentes fins telles que la prédiction et l'évaluation de performance, la sûreté de fonctionnement,... La plupart du temps des hypothèses restrictives sont considérées pour réduire la complexité du modèle. Cependant les hypothèses doivent être suffisamment réalistes pour traduire le plus possible le comportement du système. Ainsi, développer un modèle revient à faire un compromis. En effet, un modèle simple risque de manquer de précision alors qu'un modèle précis et fidèle est complexe.
Par conséquent, le processus de modélisation repose sur des hypothèses qui satisfont deux principales conditions qui sont la simplicité (afin de simplifier l'évaluation et accélérer la résolution du modèle) et le réalisme (afin de s'approcher au système réel).

3.1.2 Formalismes de modélisation

Les systèmes réels sont souvent complexes et composés par des modules qui interagissent entre eux. Afin de décrire un tel comportement, plusieurs formalismes ont été proposés.
Chacun a sa spécificité et sa manière de description des états du système et les transitions possibles entre eux. Aujourd'hui, il existe pas mal de formalismes qui permettent de relaxer les hypothèses et de développer des modèles plus généraux et plus complexes.

3.1.2.1 Chaîne de Markov

Les travaux les plus anciens ont analysé les processus stochastiques et plus généralement des chaînes de Markov. Les chaînes de Markov ont été inventées en 1913 par le mathématicien russe Andreï Markov. Une chaîne de Markov est un processus de Markov à temps discret, ou à temps continu et à espace d'états discret. Le processus de Markov est un processus aléatoire qui permet de prédire l'état futur en se basant seulement sur l'état présent et non des états antérieurs. On dit que le système n'a pas de "mémoire". Les états du système sont représentés par des variables aléatoires $(X_n)_{n\in N}$. Les transitions entre les états sont données par une matrice stochastique $Q(X_n, X_n + 1))$ représentant la dynamique du système. Cette matrice est toujours représentée par un graphe orienté composé par des nœuds et des flèches. Les nœuds représentent les différents états possibles et les flèches représentent les transitions entre les états. Une flèche allant de l'état *i* vers l'état *j* ayant un poids strictement positif $Q(i,j)$ indique la probabilité de passage à l'état *j* sachant que la chaîne soit dans l'états *i*. Pour une large classe de systèmes, les chaînes de Markov représentent un formalisme efficace pour analyser leurs performances et appréhender leurs comportements. Cependant ce type de formalisme est souvent limité lorsque l'espace d'état est important. Cette limitation est illustrée principalement par un problème de passage à l'échelle induisant des difficultés au niveau de la résolution de la chaîne et au niveau du calcul des performances.

3.1.2.2 les files d'attente

Définition
Les réseaux des files d'attente constituent un formalisme de modélisation, largement utilisé pour

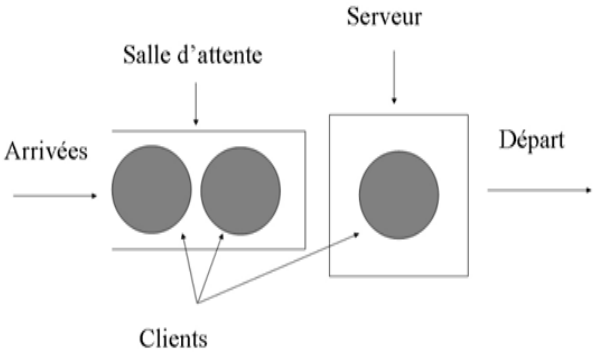


FIGURE 3.1 – File d'attente simple à un serveur

l'évaluation des performances des systèmes tels que les systèmes informatiques, les réseaux de communication et les systèmes de production. Un système d'attente consiste en une file d'attente (salle d'attente), appelée aussi buffer ou tampon de capacité finie ou infinie, d'un ou plusieurs serveurs, et d'un flux d'arrivée de clients ou travaux (jobs), générées par un processus d'arrivée externe. Les arrivées rejoignent la file pour solliciter un service offert par l'un des serveurs. Quand les serveurs sont occupés, selon la politique choisie, elles repartent ou attendent en file puis quittent le système une fois servies. L'ordre de service d'un client est défini par une politique de service.

Caractéristiques d'une file d'attente Un système de file d'attente est caractérisé par :

- *Flux d'arrivée :* Les arrivées ou les clients qui entrent dans le système pour recevoir un service, peuvent être régulières (déterministes) ou aléatoires, individuelles ou groupées, provenir de populations différentes ou se répartir en plusieurs files. La description du temps d'arrivée des clients s'appelle processus d'arrivée. Le processus d'arrivée présente le nombre (aléatoire) d'arrivées durant un intervalle de temps. D'où on parle généralement du temps d'inter-arrivés qui est représenté par des distributions de probabilités.
- *Temps de service :*c'est le temps qui sépare le début de la fin du service d'un client. Comme le temps d'inter-arrivées, des lois de probabilité décrivant le temps de service.
- *Le nombre des serveurs :* L'organe de service peut être constitué d'un ou plusieurs serveurs disposés de diverses façons (en parallèle, en série, ...). Ils peuvent être identiques ou différents, dépendants ou indépendants les uns des autres. La plupart du temps, les serveurs sont considérés comme identiques et indépendants les uns des autres.
- *La capacité de la file :* est notée K est le nombre de clients maximal qui peut se trouver simultanément dans la file et les serveurs.
- *La discipline de service :* déterminer l'ordre d'ordonnancement des clients dans la file et leurs passages dans les serveurs, afin d'être servis. Les disciplines les plus courantes :
  - FIFO (First In First Out) : le premier arrivé sera le premier servi. L'ordre de service est le même de celui d'arrivée,
  - LIFO (Last In First Out) : le dernier arrivé sera le premier servi. Chaque arrivée sera placée en tête de file pour être servi en premier,
  - PS (Processor Sharing) : Les clients sont servis à tour de rôle. Chaque client passe un quantum de temps très petit dans le serveur et revient dans la file d'attente jusqu'à la terminaison totale de son service.
  - Random : Chaque client accède aux serveurs dans un ordre aléatoire,
  - etc.

Notation de Kendall

Un modèle de file d'attente est décrit par la notation de Kendall. Cette notation introduit les paramètres est A/B/C/K/N/D où.

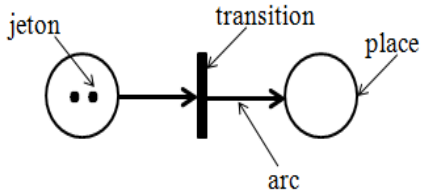


FIGURE 3.2 – Exemple d'un réseau de Petri composé de : deux places,une transition, deux arcs et deux jetons qui circulent de gauche à droite

- **A** : la distribution d'arrivée des clients.
- **B** : la distribution du temps de service (la durée passée par un client dans le serveur).
- **C** : nombre de serveurs.
- **K** : La capacité de la file.
- **N** : taille de la population des clients.
- **D** : La discipline de service.

A et **B** : représentent le processus d'arrivée et de service qui décrivent la manière dont les clients pénètrent et quittent le système. Ils peuvent être donnés par plusieurs types de distributions (loi Markovienne, Loi générale, Loi de Erlang,...).

3.1.2.3 Réseaux de Petri

Du fait de la complexité de plus en plus forte des systèmes, des travaux ont été faits pour développer des nouvelles approches de modélisation qui donnent plus de précision dans la description des comportements complexes et modélisent les différentes propriétés telles que la synchronisation, le parallélisme, les conflits, etc. Un grand nombre de ces approches a adopté les réseaux de Petri. Les réseaux de Petri ont été introduits dans la thèse de Carl Adam Petri au début des années soixante [79]. Des travaux ultérieurs ont développé les réseaux de Petri comme un formalisme mathématique particulièrement adapté à la modélisation des systèmes à événements discrets distribués et concurrents. Ce formalisme permet d'une part de modéliser les différents états du système d'une manière précise grâce à sa représentation graphique. D'autre part, il fournit une représentation mathématique permettant d'établir les équations d'état ainsi l'évaluation des performances.

Définition

Un réseau de Petri (RdP) est un outil graphique et mathématique décrivant les relations entre des conditions et des événements. C'est un graphe orienté, pondéré et biparti, c'est-à-dire comprenant deux familles de sommets : les places (représentées par des cercles) et les transitions (représentées par des traits ou des rectangles). Les places représentent l'état du système. Les transitions représentent l'ensemble des événements (les actions se déroulant dans le système) dont l'occurrence provoque la modification de l'état du système. Les arcs reliant deux sommets appartenant à deux différentes familles (une place à une transition ou une transition à une place). Chaque place contient un nombre entier de jetons (ou marques) qui indique la valeur d'état associée à cette place. L'état du système est représenté par le vecteur marquage qui définit le nombre de jetons dans chaque place. Une évolution du marquage représente l'évolution de l'état (dynamique du système). Cette évolution se produit par l'occurrence d'un événement. Un événement est représenté par un franchissement d'une transition, si certaines conditions sur le marquage des places en amont sont satisfaites.

Notation

Un réseau de Petri est un 5-uplet R = <P, T, Pre, Post, M_0 > où :

- **P** est un ensemble fini de places,
- **T** est un ensemble fini de transitions,

- **Pre** : P x T ⟶N est l'application places précédentes,
- **Post** : P x T ⟶ N est l'application places suivantes
- M_0 : P ⟶ N est l'application marquage initial.

Le franchissement d'une transition correspond à enlever les jetons dans les places d'entrée des transitions pour les mettre dans les places de sortie des transitions.

Extensions des réseaux de Petri

Des nombreuses extensions ont été ajoutées au formalisme de base de RdP telles que :

- les réseaux de Petri stochastiques (SPN) où le temps de séjour dans différents états est aléatoire (suit une loi exponentielle). Le système génère des chaînes de Markov sous jacentes ;
- les réseaux de Petri stochastiques généralisés (GSPN) ont deux classes de transitions : les transitions immédiates et les transitions temporisées. Une fois activées, les transitions immédiates se déclenchent en un temps zéro. Les transitions temporisées se déclenchent après un temps d'activation aléatoire, réparti de façon exponentielle, comme dans le cas des SPN. L'analyse du réseau se fait aussi par l'étude de la chaîne de Markov sous-jacente ;
- les réseaux de Petri de haut niveau (réseaux colorés RdPC) dans ce type du réseau, les jetons sont dits colorés où chacun a une identité propre. Un des problèmes traités par le RdPC est l'utilisation d'une ressource commune par plusieurs utilisateurs où le traitement de ce problème avec des réseaux de Petri ordinaires engendre un nombre important de structures redondantes alourdissant ainsi l'analyse.

3.1.2.4 Les SRN (Stochastic Reward Nets)

Le formalisme SRNs (Stochastic Reward Nets) est une variante des réseaux de Petri stochastiques généralisés. Il permet de structurer des modèles complexes en modules et de capturer tous les détails correspondants et les interactions entre ces modules de manière concise. Il capture explicitement des caractéristiques quantitatives et qualitatives telles que la concurrence et la synchronisation. Les SRN augmentent considérablement le pouvoir de modélisation du PN en ajoutant des fonctions de garde et en indiquant les multiplicités d'arc dépendantes, les priorités de transition et les taux de récompense. Une fonction de garde est une fonction booléenne associée à une transition. Lorsque la transition satisfait toutes les conditions d'entrée dans un marquage M, la garde est évaluée. Si la valeur de la fonction de garde est vraie, la transition est activée. Les multiplicités d'arc dépendant du marquage permettent soit d'activer le nombre de jetons requis pour la transition, soit de retirer le nombre de jetons du lieu d'entrée, soit de placer le nombre de jetons placés dans un lieu de sortie en fonction du marquage actuel du PN. Ces arcs sont appelés arcs à cardinalité variable.

3.1.2.5 Les algèbres CCS

Le Calcul des Systèmes de Communication (CCS - Calculus of Communicating Systems) a inspiré d'autres approches de modélisation. Ceci est le cas de :

- des algèbres des processus stochastiques sont un formalisme mathématique pour la description et l'étude des systèmes concurrents. Ce formalisme permet de modéliser les délais donc il est bien adapté à l'évaluation des performances. L'algèbre de processus est un langage défini par une syntaxe et une sémantique. Plusieurs variantes existent aussi pour les algèbres de processus stochastiques tels que PEPA, EMPA et TIPP.
- des réseaux d'automates stochastiques (SAN - Stochastique Automata Networks) sont basés sur une approche par composants (automates) modélisant aussi les délais. Un automate est défini par des états, transitions et événements. Chaque automate stochastique représente un sous-système. Les événements sont deux types locaux et synchronisants. Les événements locaux changent l'état local d'un seul automate tandis ceux synchronisants changent l'état local de plusieurs automates simultanément.

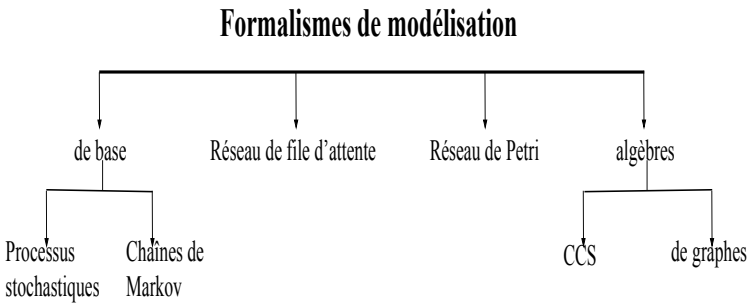


FIGURE 3.3 – Formalismes de modélisation

3.1.2.6 Les algèbres de graphes

Afin de décrire le comportement temporel d'événements, des outils algébriques ont été développés sur les graphes. Ces outils offrent une vision très distincte. Ils s'intéressent au spectre des matrices que l'on peut associer à un graphe. On peut citer les algèbres (min,+) ou (max,+) qui utilisent la théorie et l'algorithmique des graphes. Les algèbres exotiques en général ne sont pas basées sur les chaînes de Markov. Tandis que, les autres formalismes sont en majorité basés sur des hypothèses Markoviennes.
La figure 3.3 présente un résumé de la classification présentée.

3.2 Résolution du modèle

Après la mise en place du modèle, l'étape suivante est le choix d'outils permettant la prédiction et l'évaluation des performances. Dans cette section, nous sommes intéressés par le calcul de l'état stationnaire du modèle qui nous permet par la suite de calculer les différents indices de performance tels que le nombre moyen de clients dans le système, le temps moyen de séjour, etc.

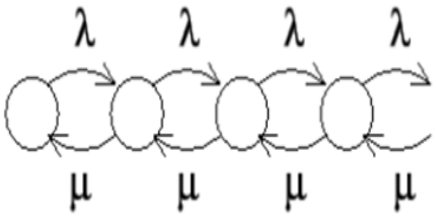


FIGURE 3.4 – Processus de naissance et mort

L'état stationnaire est exprimé par un vecteur de la probabilité stationnaire $\pi = (\pi_i)$ où chaque composant représente la proportion du temps passé dans chaque état i de l'espace d'états de la chaîne de Markov sur une durée infini. L'obtention du vecteur stationnaire correspond à la résolution du système d'équation :

$$\begin{cases} \pi Q = 0 \\ \pi e = 1 \end{cases}$$

avec
Q est la matrice générateur infinitésimal.
e est le vecteur unitaire.
Afin d'obtenir le vecteur stationnaire, plusieurs méthodes existent. Ces méthodes sont divisées en trois groupes :

- Les méthodes analytiques
- Les méthodes numériques
- Les simulations

3.2.1 Les méthodes analytiques

Les méthodes analytiques sont les méthodes qui permettent de calculer le vecteur de probabilité stationnaire sans résoudre numériquement les équations du système linéaire. Pour un système avec un grand espace, les méthodes analytiques représentent un avantage qui permet d'éviter la résolution du système linéaire [80], [81], [82]. Parmi ces méthodes on peut citer :

3.2.1.1 Processus de Naissance et Mort

Un des processus de Markov le plus connu et le plus étudié qui utilise les méthodes analytiques pour calculer les différentes probabilités stationnaires c'est le processus de naissance et de mort (birth and death process). Chaque état i de la chaîne de Markov (sauf le premier et le dernier état) possède seulement deux transitions (Fig. 3.4) :

— Une transition appelée « naissance » qui permet de passer à l'état suivant (i+1) avec un taux de naissance λ représente généralement une arrivée.

— Une transition appelée « morte » qui permet de passer à l'état précédent (i-1) avec un taux de mort μ représente généralement un départ.

3.2.2 Les méthodes numériques

Un très grand nombre de méthodes sont disponibles dans la littérature afin de résoudre les systèmes linéaires [83], [84]. Il faut bien noter que la matrice génératrice infinitésimale d'une chaîne de Markov doit avoir des caractéristiques spécifiques permettant d'appliquer ces méthodes. Les méthodes numériques peuvent être divisées en trois groupes :

— les méthodes numériques directes ;

— les méthodes numériques itératives avec analyse structurelle ;

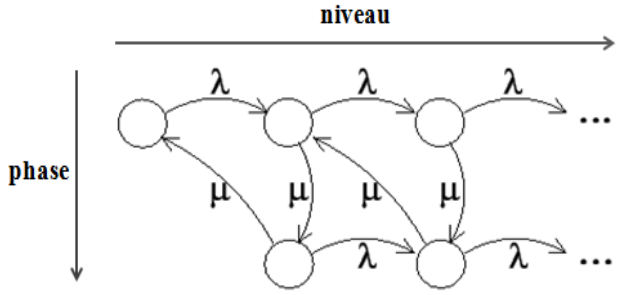


FIGURE 3.5 – Exemple d'une chaine de Markov de processus QBD

— les méthodes numériques itératives pures.

Les méthodes numériques directes comme la méthode de Gauss [85] sont utilisables pour des systèmes avec un nombre d'état réduit. Généralement, on utilise les méthodes numériques itératives pour résoudre les systèmes linéaires applicables aux chaînes de Markov. On peut trouver des méthodes simples comme la méthode de Jacobi, Gauss-Siedel et sur-relaxation successive, et des méthodes plus complexes comme les méthodes de projection (Arnoldi, GMRES, Lanczos, etc). Des définitions et des descriptions précises sont trouvées dans les livres de Saad [84].

3.2.2.1 Les méthodes numériques structurelles : Matrice Géométrique

La méthode basée sur les processus de quasi naissance et de mort (QBD - quasi birth and death process)représente une méthode numérique performante pour résoudre les systèmes avec un très grand espace d'état.

Définition

Le processus QBD est une généralisation du processus de naissance et de mort. Il représente un des outils mathématiques les plus importants pour résoudre les systèmes d'attente, systèmes de production, réseaux de télécommunication et autres systèmes. Ce type de processus a été étudié pour la première fois par Wallase dans sa thèse de Ph.D [86], après des discussions détaillées ont été présentées par Neuts sous le nom de solution de matrices géométriques [87], et par Latouche et Ramaswami [88].

Cette technique est appliquée sur une structure particulière. L'idée de cette méthode est de ne pas considérer la chaîne de Markov état par état mais bloc par bloc où chaque bloc présente un macro-état. Chaque macro-état est composé d'un sous-ensemble d'états l(i) représentant un niveau i (figure 6). Un niveau est composé de m phases. Le processus QBD à espace d'état bidimensionnel est défini comme suit :

$$\Omega=\{(i,j),i\geq,1\leq j\leq m\}$$

L'état (i,j) représente le niveau et la phase du processus, On note

— l(i) le sous-ensemble d'états représentant le niveau i,

— m est le nombre de phase dans le niveau l(i),

Chaque niveau peut avoir un nombre fini ou infini de phases m. La figure 3.5 illustre un exemple d'une chaîne de Markov d'un processus QBD.

Les transitions entre états sont limitées seulement aux états qui sont dans le même niveau ou entre deux niveaux adjacents.

Résolution / générateur infinitésimal

Les processus QBD ont des propriétés structurelles intéressantes qui peuvent être utilisées pour simplifier le calcul des probabilités stationnaires. En effet, le calcul de cette dernière a fait l'objet de nombreuses recherches. La méthode la plus connue a été proposée par Wallase [86]et généralisée après par Neuts [87]. Cette technique est appliquée sur une structure particulière où le

système est subdivisé en infinité des niveaux dont chaque niveau est composé de m phases. Ainsi, le générateur infinitésimal du processus QBD est divisé en blocs Bi. La matrice est une matrice tri-diagonale qui est représenté comme suit :

$$Q = \begin{pmatrix} B_1 & B_0 & 0 & 0 & 0 & 0 & ... \\ B_2 & A_1 & A_0 & 0 & 0 & 0 & ... \\ 0 & A_2 & A_1 & A_0 & 0 & 0 & ... \\ 0 & 0 & A_2 & A_1 & A_0 & 0 & ... \\ 0 & 0 & 0 & A_2 & A_1 & A_0 & ... \\ ... & ... & ... & ... & ... & ... & ... \end{pmatrix}$$

où B_0 ; B_1 ; B_2 ; A_0 ; A_1 ; A_2 sont des matrices carrées d'ordre m.
Pour trouver la probabilité stationnaire, il faut chercher la probabilité stationnaire pour chacun des blocs en fonction des probabilités de chacun de ses blocs adjacents. D'où le vecteur de probabilité stationnaire peut être écrit sous la forme :
$\pi = (\pi_0, \pi_1, \pi_2, ...)$, avec $\pi_i = (\pi_{i1}, \pi_{i2}, ..., \pi_{im})$ pour i=0,1,...
Dans la littérature, il existe plusieurs méthodes de résolution, on peut citer la méthode de la matrice géométrique. Cette méthode a été expliquée par Neuts dans [87]. Supposant que le processus QBD est irréductible et ergodique, il existe alors une matrice R d'ordre m tel que la distribution stationnaire est donnée comme suit : $\pi_i = \pi_0 * R^i$, i≥1. Ainsi, la solution du système π possède une forme de matrice géométrique. La matrice R est appelée "matrice de taux R" qui représente le taux de visite prévu au niveau l(i+1).
La méthode géométrique est appliquée à un espace d'états infini. Cependant, pour les processus QBD avec un espace d'états fini, la situation est tout à fait différente en raison de la présence d'états limites (de bornes) supplémentaires dans le générateur infinitésimal.

$$Q = \begin{bmatrix} B_1 & B_0 & 0 & 0 & 0 & 0 & ... \\ B_2 & A_1 & A_0 & 0 & 0 & 0 & ... \\ 0 & A_2 & A_1 & A_0 & 0 & 0 & ... \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & A_2 & A_1 & A_0 & ... \\ 0 & 0 & 0 & 0 & A_2 & A_1 & C_0 \\ 0 & 0 & 0 & 0 & 0 & C_2 & C_1 \end{bmatrix}$$

Ainsi, il n'est pas possible de garantir que la distribution stationnaire ait **une structure géométrique matricielle de la forme de l'équation (1)**. Plusieurs méthodes ont été proposées dans la littérature pour résoudre les probabilités stationnaires de processus QBD avec un espace d'états fini. L'idée commune de ces méthodes est de réduire le système d'équations de la balance globale à un système d'équations plus petit et d'exprimer les probabilités stationnaires en fonction de la solution à ce système réduit. Cependant, ces méthodes diffèrent par la façon dont le système réduit est obtenu ainsi que le calcul des probabilités en régime permanent. De plus, chaque méthode a supposé des hypothèses différentes de l'autre. Hajek [89] a proposé une méthode du calcul de vecteur de probabilité en troquant l'espace d'états infini. Cette méthode consiste à chercher la distribution de probabilité stationnaire sur les états de bornes en résolvant deux équations quadratiques matricielles pour calculer R et G. Ceci implique l'utilisation des techniques numériques. L'auteur a montré que le vecteur de probabilité stationnaire peut être écrit comme une somme de deux termes matriciels plus un terme linéaire.
Gun [90] a fournit des résultats pour les processus QBD à un espace d'états fini en appliquant ses résultats dans les files d'attente à capacité finie avec une distribution phase-type. Une expression de forme fermée (closedform) a été présentée pour les probabilités stationnaires ainsi pour la matrice R. Cette méthode s'applique aux processus QBD finis, à condition que certaines matrices soient non singulières et que certaines égalités matricielles soient valables. La solution fournit des relations entre les probabilités stationnaires qui ne peuvent pas être exploitées pour donner

une solution récursive pour les composantes du vecteur de probabilité stationnaire. Vittoria [91] propose une solution analytique matricielle explicite pour une large classe de processus de QBD finis. Une équation récursive est fournie pour le calcul des sous-vecteurs non-boundary. La seule hypothèse de cette méthode est que la matrice B est non singulière. Cette solution a été étendue à un processus QBD généralisé (GQBD). Essia H. Elhafsi et Mart Molle [92] ont proposé une solution moins complexe que les autres solutions et peut être appliquée à plusieurs systèmes de files d'attente. Cette solution traite le problème de stabilité sous certaines conditions. En résolvant ensemble des équations, les auteurs ont montré que le vecteur de probabilité stationnaire a une représentation géométrique matricielle. Ils ont proposé une expression de forme fermée pour un calcul efficace des probabilités stationnaires où la matrice géométrique dépend du nombre de niveaux.

3.3 Conclusion

La modélisation des systèmes est une étape fondamentale qui permet d'analyser les systèmes, prédire et évaluer les performances des systèmes. Dans ce chapitre nous avons présenté un état de l'art sur la modélisation stochastiques. Nous avons rappelé les différents formalismes de modélisations ainsi nous avons introduit les méthodes de résolutions utilisées afin d'obtenir les mesures de performances. Dans les chapitres suivants, ces méthodes seront exploitées pour étudier le problème de vieillissement de logiciel dans les systèmes virtualisés, ainsi que pour améliorer les performances des équipements réseaux dans les centres de données.

Chapitre 4

Contribution 1 : Performabilité et gestion de l'énergie pour les systèmes de serveurs virtualisés (SVSs)

La contribution, sur laquelle nous nous focalisons dans ce chapitre, concerne la modélisation et l'évaluation de la performabilité d'un SVS utilisant le rajeunissement logiciel comme méthode proactive pour faire face aux aléas du vieillissement logiciel. Cette modélisation intégrera aussi une politique de gestion dynamique de l'énergie d'un PMC (Power Manageable Component) possédant des états soft multiples (P-state et C-state). Elle sera basée sur une approche modulaire utilisant les SRN (Stochastic Reward Nets) et décrivant les états d'énergie du PMC et les différentes transitions régissant le passage d'un état à l'autre. Dans ce chapitre, nous présentons dans un premier lieu une synthèse bibliographique des travaux existants qui traitent le phénomène de vieillissement logiciel et de rajeunissement. Nous décrivons après notre modèle proposé et ses différents paramètres. Par la suite, nous présenterons nos résultats numériques.

Sommaire

4.1 Travaux annexes

Cette section passe en revue des publications et des études qui analysent le problème du vieillissement. L'analyse du vieillissement repose soit sur une approche basée sur modélisation, sur la mesure, ou hybride.

4.2 Analyse de vieillissement

4.2.1 Analyse de vieillissement basant sur la modélisation

4.2.1.1 Formalisme de modélisation

Le processus de vieillissement dans les systèmes virtualisés (avec une ou plusieurs machines virtuelles, hôte physique et diverses stratégies de rajeunissement) est modélisé par plusieurs types de formalismes stochastiques, y compris modèles d'espace d'états tels que les Réseaux de Petri Stochastiques (SPN : stochastic Petri nets), les Réseaux de Récompense Stochastiques (SRN : stochastic Rewards nets), chaînes de Markov à temps continu (CTMC : continuous-time Markov chains), processus semi-markov (SMP : semi- markov processes),... et modèles combinatoires - tels que les diagrammes de fiabilité (RBD : reliability block diagrams) et les arbres de défaillances dynamiques (DFT : dynamic fault trees).

— **Modèles basés sur SPN :** les auteurs Xu et al. [93] et Rezaei et Sharifi [94] ont utilisé formalisme SRN pour modéliser un système virtualisé de serveurs avec un rajeunissement basé sur le temps appliqué au niveau VMM et un rajeunissement basé sur les mesures au niveau des VM. Nguyen et al. [95] ont utilisé aussi SRN pour étudier différents modes de défaillance et de récupération de plusieurs VM et VMM. Trois politiques de rajeunissement (pas de rajeunissement, rajeunissement basé sur le temps et rajeunissement basé sur le temps et la charge) ont été proposées dans [96] appliquées à un système virtualisé de serveurs avec plusieurs VM sur un seul VMM. Machida et al. ([97],[98]) ont proposé des modèles de disponibilité de trois techniques de rajeunissement pour un système virtualisé de serveurs avec rajeunissement basé sur le temps, aussi bien pour les VMs que le VMM. MRSPN (Markov Regenerative Stochastic Petri Nets) est une autre variante de SPN qui a été adoptée par Okamura et al [99] pour présenter une analyse transitoire des deux techniques de rajeunissement Cold- VM et Warm-VM.

— **CTMC :** Myint et Thein [100] ont présenté un modèle de disponibilité multi-hôtes multi-VM basé sur CTMC. Ce modèle inclut un équilibrage de charge et un mécanisme de rajeunissement pour chaque VM. Thein et al. [101] ont présenté un modèle de disponibilité d'un système virtualisé avec une politique de rajeunissement basée sur le temps. Une configuration de cluster a été étudiée pour montrer la performance du modèle (un seul serveur et deux serveurs qui hébergent plusieurs VM). Les auteurs dans [102] ont présenté un framework de rajeunissement logiciel qui offre une haute disponibilité aux systèmes de serveurs d'applications. Ils ont proposé une CTMC pour modéliser un seul hôte avec plusieurs VM.

— **Modèles combinatoires :** RBD et DFT sont présents dans plusieurs travaux pour traiter le phénomène de vieillissement logiciel. Citons, par exemple, les travaux de Melo et al. [103] qui ont proposé un modèle de disponibilité avec une technique de rajeunissement Migrate-VM. Ce travail est basé sur un RBD étendu et des réseaux de Petri stochastiques déterministes (DSPN). Aussi, on peut citer les travaux de Rahme et al. qui exploitent DFT pour modéliser le rajeunissement de logiciels dans le Cloud [104].

— **Les processus semi-markoviens :** Machida et al. ont utilisé le processus semi-markovien afin d'analyser le temps d'exécution des tâches exécutant dans des serveurs virtualisés. Ces derniers sont sujets du vieillissement et rajeunissement du VMM ([105],[106]).

— **Modèle d'optimisation :** c'est un modèle hybride pour analyser et atténuer les effets du vieillissement. Wu et al. ont proposé un modèle dans le but de réduire la consommation d'énergie et les retards de transmission dans les appareils mobiles affectés par le vieillissement [107].

4.2.1.2 Métriques

Les analyses basées sur la modélisation s'intéressent principalement par des mesures de fiabilité, dont la plus connue est la disponibilité, performances, dégradation des performances. Les principales métriques de performabilité sont discutées dans la littérature sont présentées ci-dessous.

— **La disponibilité** c'est la métrique la plus investiguée dans les analyses basées sur la modélisation. Elle revêt une importance particulière pour les environnements cloud. Xu et al. [93], Rezaei et Sharifi [94], Nguyen et al. [95] et Melo et al. [103] modélisent la disponibilité du cloud en utilisant le formalism SPN sous deux stratégies de rajeunissement basées sur la migration (avec et sans test avant la migration). Thein et al. [102] analysent la disponibilité du cloud avec différentes configurations de cluster (2 VM sur le même serveur et 2 VM chacune sur un serveur). Les auteurs ont proposé une politique de rajeunissement basée sur le temps nommée VMSR (Virtual Machine based Software Rejuvenation) pour les serveurs d'applications. Ils ont montré que cette politique assure une meilleure disponibilité et optimise les performances d'applications vieillissantes.

— **Les métriques de dégradation des performances**, telles que le temps de réponse, le temps d'exécution des tâches, le débit, sont présentes dans plusieurs travaux basés sur la modélisation. Machida et al. [105] ont proposé un modèle pour analyser le temps de réalisation des travaux, tandis que dans [107] et [108], les modèles ont été conçus pour évaluer la performance et la consommation d'énergie. D'autres travaux examinent l'impact du vieillissement et rajeunissement sur la disponibilité, la fiabilité, la survivabilité et la performance. A titre d'exemple, les travaux de Nguyen [95] ont évalué la perte de transaction et la disponibilité. Le travail de Machida [106] a étendu l'analyse du temps de travail à l'analyse de la disponibilité à l'état stable.

4.2.2 Analyse de vieillissement basant sur des mesures

Les analyses basées sur des mesures utilisent les données collectées en surveillant des indicateurs de vieillissement afin de déduire ou prédire l'état de vieillissement d'un système. Ces indicateurs peuvent être mesurés comme la consommation des ressources système, par exemple la mémoire et le stockage, ou perçues par l'utilisateur, telles que le temps de réponse, la latence et le débit. Ces données sont collectées au cours de l'exécution et sont utilisées pour prévoir les défaillances futures. En se basant sur des observations passées, on prend des décisions sur le rajeunissement, ainsi que sur l'équilibrage de charge et l'allocation de ressources statiques ou dynamiques. Plusieurs techniques sont utilisées comme l'analyse de séries chronologiques, l'apprentissage automatique, les approches basées sur des seuils, et autres.

4.2.3 Techniques d'analyse hybride

Les techniques hybrides présentent une généralisation des deux méthodes précédentes. Elles sont proposées comme une combinaison de solutions basées sur la modélisation et sur les mesures. Les solutions hybrides adoptent un modèle stochastique pour décrire le phénomène et déterminent les paramètres du modèle au moyen de mesures, c'est-à-dire de données d'observation. Le travail de Liu et al. [109] représente un exemple d'une technique hybride. Les auteurs mesurent diverses ressources (CPU, stockage, réseau) en se basant sur un système de streaming. Ces mesures ont été utilisées pour paramétrer un modèle de file d'attente afin de planifier le rajeunissement.

4.3 Corrélation entre vieillissement logiciel et charge de travail

Plusieurs études ont montré qu'il existe une forte corrélation entre la charge de travail et le vieillissement [110], [111], puisque le taux d'épuisement des ressources dépend de la charge effective du système. Bruneo et al. ont proposé une technique permettant d'évaluer le processus de vieillissement de VMM et de trouver la politique de rajeunissement optimale basée sur la charge de travail qui maximise la disponibilité de VMM. Ils exploitent la théorie de la fiabilité dynamique et des techniques algébriques symboliques. La solution proposée par Kadirvel et Fortes [112] est appliquée à un gestionnaire de système pour gérer et contrôler les ressources virtualisées afin de prendre des des décisions (telles que l'augmentation et la diminution de la capacité des ressources, migrations de VM, modifications dynamiques de la configuration des ressources). Cette solution combine une approche basée sur un réseau de Petri pour modéliser le gestionnaire de système, souffrant de vieillissement, avec l'utilisation de la théorie du contrôle pour contrôler la consommation de ressources. Zhao et al. [113] ont représenté les différents objectifs d'un fournisseur de services (qui veut maximiser la disponibilité) et d'un mainteneur (qui veut minimiser les coûts) en utilisant la théorie de jeu et les processus de renouvellement de Markov pour décrire les états de dégradation du système et déterminer le programme de rajeunissement optimal.

4.4 Analyse de Rajeunissement logiciel

Machida et. al [98] ont proposé un modèle de disponibilité de trois techniques de rajeunissement pour un système virtualisé de serveurs avec rajeunissement basé sur le temps, aussi bien pour les VMs que le VMM en utilisant les SRN. Ils ont étudié la disponibilité des VMs en régime permanent et le nombre de transactions perdues. Les résultats montrent que le rajeunissement Migrate-VM réalise la meilleure disponibilité, en raison de la possibilité de préserver l'exécution des VMs même pendant le rajeunissement VMM. De plus, les auteurs ont constaté que l'utilisation de la migration pré-copie, ainsi que l'utilisation de la politique de retour après rajeunissement sont très efficaces. Kourai et al. [114] ont présenté une technique appelée VMBeam permettant le rajeunissement logiciel léger des systèmes virtualisés grâce à la migration sans copie. Lors un rajeunissement d'un ancien système virtualisé, VMBeam démarre un nouveau système virtualisé sur le même hôte à l'aide de la virtualisation imbriquée. Ensuite, il migre tous les VMs du vieux système virtualisé vers le nouveau en déplaçant la mémoire des VMs, sans aucune copie. Une approche proposée par Torquato et al. [115] basée sur des expériences et des observations empiriques pour étudier le vieillissement et le rajeunissement Migration directe sur des systèmes logiciels. Comme VMM, OpenNebula et KVM ont été utilisés. Torquato et al. [116] ont présenté des modèles de disponibilité basés sur les réseaux de Petri stochastiques pour évaluer deux approches de migration VM :Warm-Standby et Cold-Standby. Ces approches sont basées sur les schémas de redondance. La comparaison de la disponibilité entre les deux approches révèle que l'approche Cold-Standby a un résultat meilleur en raison de la diminution du nombre total de migrations VM. Les résultats de consommation d'énergie montrent aussi que l'approche Cold-Standby est plus efficace en termes de consommation d'énergie.

4.5 Contributions

Notre contribution dans ce chapitre porte sur l'analyse de la performabilité d'un SVS avec gestion efficace des ressources énergétiques. La problématique que nous abordons dans ce chapitre peut être décrite à partir des questions suivantes :

— Comment à partir d'une modélisation formelle modéliser les différentes entités représentant respectivement un SVS à savoir : le VMM, les VMs et le PMC (Power Manageable Component) ?

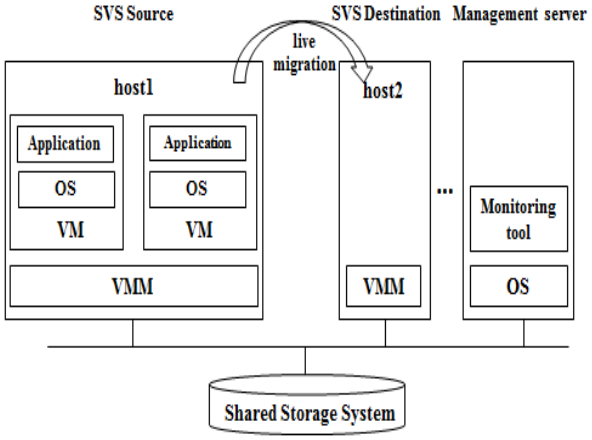


FIGURE 4.1 – Architecture d'un système virtualisé de serveur

— Comment implémenter une politique de gestion dynamique de l'énergie et l'intégrer dans un modèle de disponibilité d'un SVS ?

— Comment définir des métriques de performabilité à partir du modèle du SVS mettant en évidence l'efficacité énergétique, la haute disponibilité et les compromis nécessaires énergie-performance ?

4.6 Description du modèle versatile SVS

Nous nous focalisons dans ce chapitre sur la modélisation et l'évaluation de la performabilité d'un SVS utilisant le rajeunissement logiciel comme méthode proactive pour faire face aux aléas du vieillissement logiciel. Cette modélisation intégrera aussi une politique de gestion dynamique de l'énergie d'un PMC possédant des état soft multiples (P-state et C-state). Elle sera basée sur une approche modulaire utilisant les SRNs et décrivant les états d'énergie du PMC et les différentes transitions régissant le passage d'un état à l'autre. Le formalisme SRN présente plusieurs avantages permettant de développer des modèles concis grace à ses caractéristiques (gardes, priorités, transitions avec des distributions stochastiques dépendantes ou non des marquages).
Le modèle proposé est une extension du modèle proposé dans [117] où les auteurs ont fait recours à une abstraction du sous modèle de disponibilité SVS afin de contourner la complexité du modèle versatile SVS et de réduire l'espace d'état correspondant. Cette abstraction suppose que le modèle de disponibilité SVS se comporte comme une boîte noire commutant entre trois états (i.e. up (_vup), rejuvenate (P_vrej) et stop (P_vstop)). Ces états peuvent être classifiés en deux types. Le premier inclut l'état up et correspond à un système VMM-VM opérationnel et donc disponible ce qui permet d'exécuter des applications. Le deuxième type inclut les états stop et rejuvenate et décrit un système VMM-VM non disponible où les applications sont stoppées. Cependant, dans notre travail, nous avons conservé les spécifications détaillées. Ceci est principalement utile pour des investigations numériques plus concrètes. Aussi, les auteurs ont considéré seulement la technique de rajeunissement Cold-VM. Par contre dans ce travail, nous avons traité et comparé deux techniques "Cold-VM" et "Migrate-VM".
Dans la suite de cette section nous présenterons l'architecture d'un système virtualisé dans un premier temps. Nous détaillons les entités du modèle versatile SVS et ses attributs (places, transitions et gardes). Dans cette thèse, nous proposons deux modèles SVS, dont le premier modèle SVS est sujet au rajeunissement Cold-VM, tandis que le second utilise migrate-VM comme technique de rajeunissement VMM. Chacun est composé de VMM hébergeant une VM qui à son tour héberge des applications avec un trafic variable. Le modèle SVS est composé de deux sous-modèles dépendants :

1. Sous modèle SVS de disponibilité avec rajeunissement (Cold-VM (sous-modèle[11], Fig.4.2)ou Migrate-VM (sub-model[21],Fig.4.3)) [97] : Il modélise aussi bien le VMM que la VM et les interactions entre eux.
2. Sous modèle workload-aware PM SRN (Fig.4.4) : Il modélise un mécanisme de gestion de l'énergie permettant d'adapter les états d'énergie du PMC en fonction de la charge du SVS. Dans ce sens, le PMC est considéré comme un système d'attente particulier soumis à un trafic sporadique (modélisant des applications de données intensives) avec un serveur possédant plusieurs états d'énergie et plusieurs débit (throughput) de service.

4.6.1 Hypothèses

Avant de décrire le modèle SVS, nous allons présenter les différentes hypothèses utilisées dans notre modèle. Comme nous l'avons décrit dans le chapitre précédent, la modélisation est basée sur des hypothèses spécifiques permettant de réduire la complexité du modèle. Cependant les hypothèses doivent être suffisamment réalistes pour se rapprocher le plus possible du comportement du système.
Notre modélisation du système SVS est basée sur un certain nombre d'hypothèses à savoir :

— le SVS est composé de VMM hébergeant un VM qui à son tour supportent des applications ;

— les VMs et le VMM sont des systèmes dégradés sujet à plusieurs épreuves le long de leur processus d'exploitation telles que le vieillissement, les défaillances et les actions de maintenance préventive ;

— le VMM et la VM utilisent le rajeunissement logiciel pour faire face au vieillissement logiciel ;

— le vieillissement logiciel dépend de la charge de travail ;

— le trafic engendré par les applications est de nature variable et sporadique et il est modélisé par un processus MMPP ;

— le PMC possède plusieurs états soft (P-states, C-states), états actifs ou opérationnels (fastactive, slowactive), état idle (idle, état sleep (sleep) et état off (off) ;

— une politique de gestion de l'énergie basée sur *timeout* sera implémentée ;

— le SVS alloue dynamiquement les ressources en fonction de la charge avec une mise à l'échelle dynamique de la capacité de traitement (service), le service est aussi modélisé par un processus MMPP ;

— on utilise des distributions Markoviennes et non-Markoviennes (Déterministe), pour décrire des transitions temporisées afin de construire des modèles assez réalistes ;

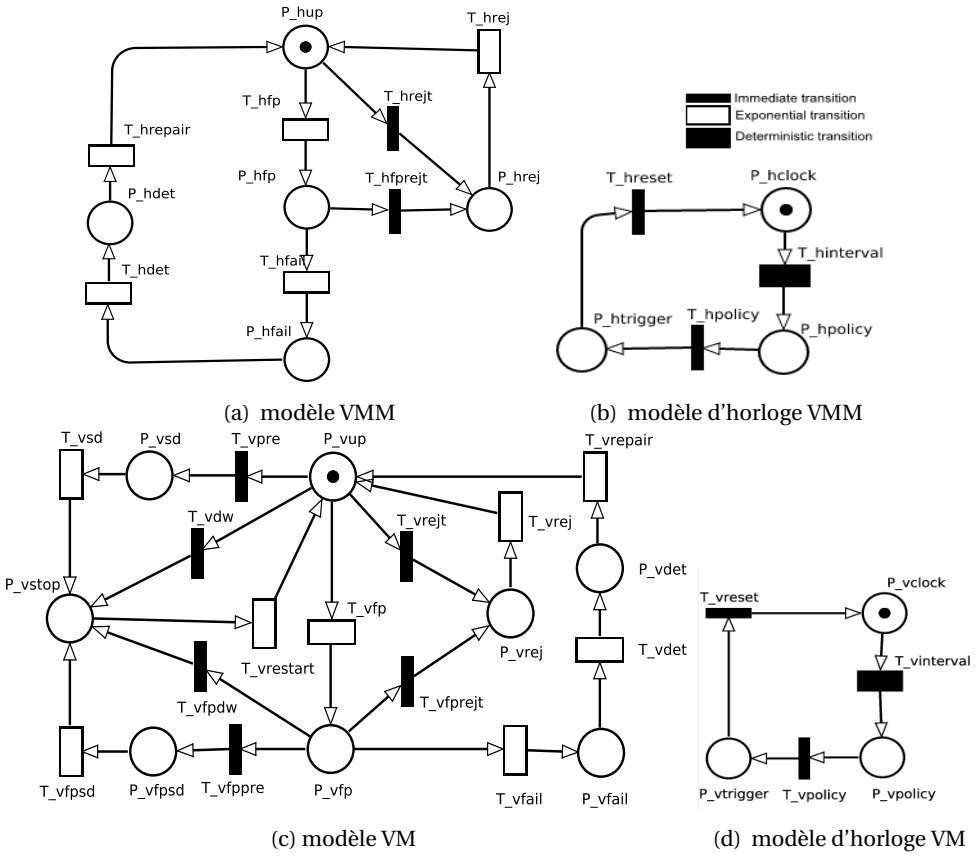
— s transitions représentant les intervalles de déclenchement de rajeunissement de la VM et du VMM sont déterministes ;

Dans la suite de cette section nous détaillons les entités du chaque modèle SVS et ses attributs (places, transitions et gardes). Nous détaillons ensuite les métriques de performabilité définis dans ce sens et les résultats numériques correspondants, mettant en évidence les compromis performance-énergie-disponibilité.

4.6.2 Modèle de disponibilité SVS avec rajeunissement

Dans cette section, nous allons rappeler le principe de fonctionnement d'un modèle de disponibilité avec rajeunissement VMM. Les deux modèles, proposés dans [112], seront exploités dans notre modèle SVS, comme un sous modèle, avec quelques modifications. Chaque modèle de disponibilité SVS avec rajeunissement (Cold-VM ou Migrate-VM) se compose de quatre sous-modèles ; *(a)* modèle VMM, *(b)* modèle d'horloge VMM, *(c)* modèle VM et *(d)* modèle d'horloge VM. Les deux modèles d'horloge sont utilisés pour le déclenchement du rajeunissement VM et VMM en fonction du temps. Le modèle VM est étroitement dépendant de celui du VMM. Ceci

évident dans la mesure où les pannes et les rajeunissements du VMM affectent automatiquement la VM hébergée. Cette dépendance entre la VM et le VMM est illustrée au travers les gardes assignées dans le modèle VM.
Notons que nous retenons les mêmes notations pour les noms des transitions et des places. Par exemple Tvfp (resp. Thfp) est une abréviation de la transition fail-prône de la VM (resp. du host (i.e. VMM)).



(a) modèle VMM (b) modèle d'horloge VMM

(c) modèle VM (d) modèle d'horloge VM

FIGURE 4.2 – Sous modèle SRN de SVS avec rajeunissement Cold-VM (sub-model[11])

4.6.2.1 Modèle de disponibilité SVS avec rajeunissement Cold-VM

Description du modèle VMM : le modèle VMM représente trois processus : *le processus d'échec* (en deux étapes), *le processus de recouvrement* (en deux étapes) et *le processus de rajeunissement* d'un VMM. La transition *T_hinterval* est de distribution déterministe. Elle est utilisée pour déclencher le rajeunissement VMM (Cold-VM) de durée 1/th après le temps de démarrage du VMM et aussitôt la transition *T_hpolicy* devient franchissable. Dès que la transition immédiate *T_hpolicy* devient franchissable (Il existe un jeton dans la place P_hup ou P_hfp et la VM est indisponible), le jeton passe de P_hpolicy vers P_htrigger et le processus de rajeunissement est déclenché. La transition immédiate *T_hrejt* ou *T_hfprejt* sont activées, et un jeton est déposé dans P_hrej. Lorsque le rajeunissement VMM nettoie les états de vieillissement, le jeton dans P_hrej est retiré et un jeton est déposé dans P_hup par franchissement de la transition *T_hrej* et un autre jeton est déposé dans P_hclock. Une fois la transition de vieillissement logiciel *T_hfp* est franchie, un jeton est déposé dans P_hfp. Cette transition représente le vieillissement logiciel du VMM. Si la transition *T_hfail* est franchie, un jeton est déposé dans la place P_hfail. Cette dernière représente l'état d'échec du VMM dû au vieillissement logiciel. *T_hdet* est franchie aussitôt la panne du VMM est détectée et *T_hrepair* devient franchissable lorsque le VMM achève la phase de recouvrement et quitte l'état d'échec. Chaque fois que le rajeunissement VMM est déclenché par le modèle d'horloge VMM.

Description du modèle VM : le modèle du VM est basé aussi sur *un processus d'échec* (en deux étapes), un *processus de récupération* (en deux étapes) et un *processus de rajeunissement* basé sur le temps. Le rajeunissement basé sur le temps est commandé par le modèle d'horloge VM. Une transition déterministe *T_vinterval* se déclenche avec une durée constante 1/tv après le dernier temps de démarrage de la VM et initie le rajeunissement tant que la transition immédiate *T_vpolicy* est activée. Lorsque le processus de rajeunissement VM se termine, la transition immédiate *T_vreset* est activée et un jeton est de nouveau déposé à la place P_vclock. Etant donné que l'exécution de la VM dépend de celle du VMM qui l'héberge, s'il n'y a pas de jeton dans P_hup ni dans P_hfp, les transitions immédiates *T_vdw* et *T_vfpdw* sont activées et un jeton est déposé dans P_vstop. *T_vrestart* est activée seulement quand il y a un jeton dans P_hup ou P_fup, et donc la VM ne peut redémarrer tant que le VMM ne redevienne disponible. Le rajeunissement Cold-VM arrête la VM hébergée avant de commencer le rajeunissement VMM. Cette politique est représentée par des fonctions de garde pour les transitions immédiates *T_vpre* et *T_vfppre*. Ces dernières sont activées lorsqu'un jeton est déposé dans P_hpolicy, dans le modèle d'horloge VMM. Dans le

TABLEAU 4.1 – Description des transitions du sous-modèle avec rajeunissement Cold-VM et du sous-modèle avec rajeunissement Migrate-VM

Description commune pour les deux sous-modèles	
Transition	**Description**
T_$hinterval$ (resp. T_vinterval)	durée d'inter-rajeunissement du (VMM) (resp. VM)
T_$hpolicy$ (resp.T_$vpolicy$)	Transitions immédiates sont activées pour déclencher le rajeunissement du VMM (resp VM)
T_$hreset$ (resp. T_$vreset$)	transitions immédiates sont activées lorsque la durée de rajeunissement de VMM (resp. VM) expire
T_hfp (resp. T_vfp)	durée de vieillissement logiciel du VMM (resp. VM)
T_$hfail$ (T_$vfail$)	Durée d'échec de VMM (resp. VM))
T_$hdet$ (resp. T_$vdet$)	durée de la détection de défaillance de VMM (resp. VM)
T_$hrepair$ (resp. T_$vrepair$)	durée de reprise après l'échec de VMM (resp. VM)
T_$hrejt$ (resp. T_$vrejt$)	transitions immédiates activées pour déclencher le rajeunissement de VMM (ou de la VM)
T_$hrej$ (resp.T_$vrej$)	durée de rajeunissement de VMM (resp. VM)
T_$hfprejt$ (resp.T_$vfprejt$)	transitions immédiates activées pour déclencher le rajeunissement de l'hôte (ou de la VM)
T_$vrestart$	Durée de redémarrage de la VM
T_vdw, T_$vfpdw$	Durée d'arrêt de la VM
Description spécifique pour chaque modèle	
Cold-VM	
Transition	**Description**
T_$vpre$, T_$vfppre$	transitions immédiates activées pour lancer le processus d'arrêt (avant le déclenchement de rajeunissement de VMM)
T_vsd, T_$vfpsd$	temps d'arrêt de la VM
Migrate-VM	
Transition	**Description**
T_$vpre$, T_$vfppre$	transitions immédiates activées pour lancer la migration de VM (avant le déclenchement de rajeunissement de VMM)
T_$vmig$, T_$vfpmig$	Durée de la migration de la VM
T_$vpost$, T_$vfppost$	Transitions immédiates déclenchant la migration de la VMM vers le VMM d'origine
T_$vbac$, T_$vfpbac$	durée de la migration de VM vers le VMM d'origine

modèle du VM, un jeton est déposé dans P_vsd ou P_vfpsd ce qui permet de déposer un jeton dans P_vstop et de tirer la transition immédiate *T_hpolicy* qui à son tour une fois franchie déclenche le processus de rajeunissement VMM. Nous supposons que le rajeunissement VMM ne peut démarrer que si un jeton est déposé dans P_vstop, P_vfail, P_vdet ou P_vrej. Notons aussi que la distribution de *T_vinterval* (resp. T_hinterval) est considérée déterministe étant donnée qu'elle est utilisée pour modéliser un intervalle de déclenchement du rajeunissement VM (resp. VMM) fixe. Il est à noter aussi que les auteurs dans [97] utilisent des distributions Erlang 10 pour approximer les distributions déterministes étant donné que l'implémentation du modèle est effectuée avec le package SPNP [53] qui utilise uniquement des distributions temporisées exponentielles. Dans la mesure ou dans notre modélisation on utilise les réseaux de Petri extended deterministic stochastic Petri nets (eDSPNs) implementés au travers l'outil TimeNet 4.1 [118] l'approximation précédente est relaxée et des distributions déterministes sont effectivement utilisées.

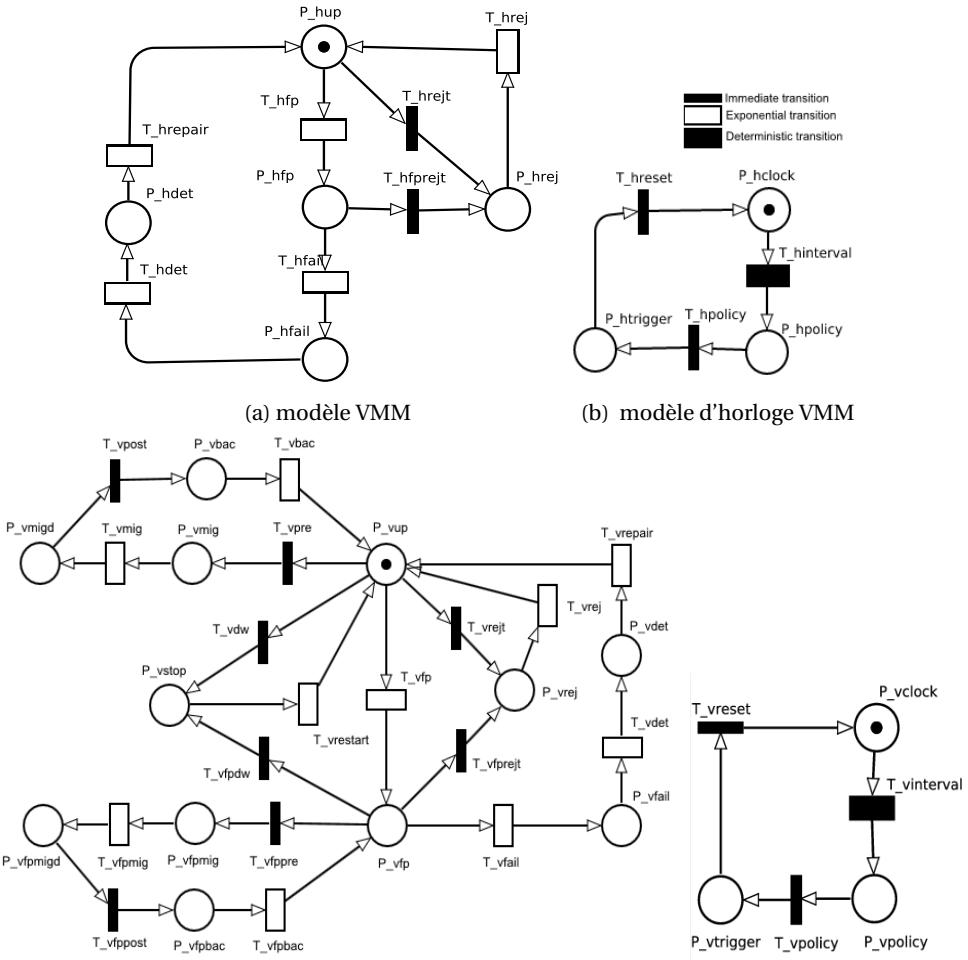
4.6.2.2 Modèle de disponibilité SVS avec rajeunissement Migrate-VM

Le modèle de disponibilité avec rajeunissement Migrate-VM comprend aussi 4 sous modèles : modèle VMM, modèle d'horloge VMM, modèle VM et modèle d'horloge VM. Le modèle VMM comporte les mêmes places que celles du modèle avec rajeunissement Cold-VM mais avec un changement de quelques gardes. Les processus de rajeunissement de VMM et VM sont basés sur le temps et se déclenchent par une horloge représentée par le modèle d'horloge. Le rajeunissement Migrate-VM utilise la migration de machines virtuelles en direct (Live Migration). Avant le déclenchement du rajeunissement VMM, on déplace la machine virtuelle en cours de travail vers un VMM distant. Si un jeton est déposé dans P_hpolicy et il existe un jeton dans P_vup ou P_vfp, les transitions immédiates *T_vpre* et *T_vfppre* sont activées et le processus de migration de VM vers un hôte distant est déclenché. Une fois le jeton est déposé dans P_vmigd ou P_vfpmigd, le processus de rajeunissement commence et l'exécution de la VM sur l'hôte distant se poursuit. Dès que le rajeunissement de VMM est complètement achevé, les transitions immédiates *T_vpost* ou *T_vfppost* seront activées et la VM retournera de l'hôte distant vers l'hôte d'origine, par migration (Live MIgration). Nous supposons qu'aucune panne ne se produit pendant les périodes de rajeunissement de VMM et de migration de VM. Le rajeunissement VM peut être déclenché, si un jeton est déposé dans P_vup, P_vfp, P_vmigd ou P_vfpmigd.

4.6.2.3 Sous modèle workload-aware PM SRN

Les deux sous-modèles workload-aware PM SRN pour les deux types de rajeunissement Cold-VM et Migrate-VM ont la même représentation mais ils ont des gardes différentes (Tab. 4.4, Tab. 4.5).
Le sous-modèle workload-aware PM est conçu pour réaliser une politique PM adaptative dynamique reposant sur un processus de suivi de la charge de travail. Le processus de suivi vise à prendre des décisions tenant compte de la charge de travail, ce qui permet de choisir judicieusement comment et quand forcer PMC à passer d'un état d'alimentation à un autre. Cela signifie que les fluctuations de la charge de travail peuvent entraîner un changement d'état PMC à tout moment. Ceci est généralement réalisé afin de maintenir de faibles coûts d'énergie tout en répondant aux exigences de performances du SLA.
Dans ce qui suit, nous donnons une description détaillée du sous-modèle de SRN PM proposé (Fig.4.4, Tab.4.4, Tab.4.5, Tab.4.7).
Hypothéses :
Les hypothèses considérées dans cet article sont principalement liées au marquage initial, à la distribution du processus d'arrivée, à la distribution du processus de service et aux transitions PMC du Sleep au wakeup. Nous supposons que :

— les distributions de transition sont soit immédiate (imm.), Exponentielle (exp.) Ou déterministe (dét.) ;



(a) modèle VMM (b) modèle d'horloge VMM

(c) modèle VM (d) modèle d'horloge VM

FIGURE 4.3 – Sous modèle SRN de SVS avec rajeunissement Cold-VM Migrate-VM (sub-model[21])

— PMC a des états d'alimentation suivants : état éteint (Poff), états actif ou opérationnel (Pfastactif, Pslowactif) et état inactif (Pidle, Pwakeup, Psleep) ;

— PMC est initialement en état de veille (Psleep) ;

— SVS est initialement inactif (#Pqueueplace = K = 3, où K est la capacité du buffer du modèle SVS) ;

— le traitement du service PMC suit un MMPP à deux états, ce qui permet de capturer la mise à l'échelle dynamique de la vitesse du PMC ;

— le traitement du service PMC dépend de la charge de travail ;

— l'arrivée de la demande suit une MMPP à deux états, ce qui permet de capturer la caractéristique du trafic en rafale ;

— il existe 4 sources de trafic (**X** sources pour un trafic élevé et **Y** sources pour un trafic faible avec **X + Y = 4**) ;

— initialement, il y a 4 sources de faible trafic ;

— PMC est initialement en état Sleep. Selon cette hypothèse et lorsqu'une demande arrive, le traitement de la demande n'est pas accordé immédiatement. Mais, il est déclenché après un délai donné (Twakeup). Ce retard correspond au temps nécessaire à PMC pour passer de l'état de Sleep à l'état actif ;

— La consommation d'énergie PMC pendant la durée de réveil Twakeup est maximale et égale à *Powerfastactive*.

TABLEAU 4.2 – Les gardes des transitions du sous modéles SVS SRN (sous-modèle[11], sous-modèle [21])

Gardes communs pour sous-modèle [11] and sous-modèle [21]	
Transitions	**Gardes**
T_hrej	*P_hclock* == 1
T_hinterval	(*P_hup* == 1) ‖ (*P_hfp* == 1)
T_hrejt	*P_htrigger* == 1
T_hfprejt	*P_htrigger* == 1
T_hreset	*P_hrej* == 1
T_vpolicy	(*P_vup* == 1) ‖ (*P_vfp* == 1)
T_vdet	(*P_hup* == 1) ‖ (*P_hfp* == 1)
T_vrepair	(*P_hup* == 1) ‖ (*P_hfp* == 1)
T_vrej	(*P_vclock* == 1) && (*P_hup* == 1) ‖ (*P_hfp* == 1)
T_vrejt	*P_vtrigger* == 1
T_vfprejt	*P_vtrigger* == 1
T_vpre	*P_hpolicy* == 1
T_vdw	*P_hfail* == 1
T_vfpdw	*P_hfail* == 1
T_vfppre	*P_hpolicy* == 1
T_vrestart	(*P_hup* == 1) ‖ (*P_hfp* == 1)
T_vreset	*P_vrej* == 1
Gardes spécifiques à chaque modèle	
Cold-VM (sous-modèle[11])	
Transitions	**Gardes**
T_hinterval	(*P_hup* == 1) ‖ (*P_hfp* == 1)
T_hpolicy	(*P_vstop* == 1) ‖ (*P_vfail* == 1) ‖ (*P_vdet* == 1) ‖ (*P_vrej* == 1)
T_hinterval	(*P_vup* == 1) ‖ (*P_vfp* == 1) ‖ (*P_vsd* == 1) ‖ (*P_vfpsd* == 1)
T_vpolicy	(*P_vup* == 1) ‖ (*P_vfp* == 1)
Migrate-VM (sous-modèle[21])	
Transitions	**Gardes**
T_hinterval	(*P_hup* == 1) ‖ (*P_hfp* == 1)
T_hpolicy	(*P_vfail* == 1) ‖ (*P_vdet* == 1) ‖ (*P_vrej* == 1) ‖ (*P_vmigd*== *1*)‖ (*P_vfpmigd* == 1)
T_vinterval	(*P_vup* == 1) ‖ (*P_vfp* == 1) ‖ (*P_vmigd* == 1) ‖ (*P_vfpmigd* == 1)
T_vpolicy	(*P_vup* == 1) ‖ (*P_vfp* == 1)

TABLEAU 4.3 – Description des transitions du sous-modèle workload-aware PM

Transition	**Description**
T_$h2l$	durée de temps pour que le trafic MMPP passe du débit élevé au débit bas
T_$l2h$	durée de temps pour que le trafic MMPP passe du débit bas au débit élevé
1/T_$hightraffic$	taux de trafic élevé
1/T_$lowtraffic$	taux de trafic bas
1/T_$servfast$	taux de service rapide
1/T_$servslow$	taux de service lent
1/T_$f2s$	taux de transition pour commuter de service rapide à lent
1/T_$s2f$	taux de transition pour commuter de service lent à rapide
1/T_$fast$	Taux de traitement rapide
1/T_$slow$	Taux de traitement lent
T_$start$	transition immédiate activée pour déclencher le traitement du service
T_$sleep2wakeup$	transition activée pour passer de l'état sleep vers l'état wakeup
T_$wakeup$	transition activée pour passer de l'état wakeup vers l'état idle
T_$timeout$	transition activée pour passer de l'état idle vers l'état sleep
T_$lossjob$	transition immédiate activée lorsqu'une requête est perdue
T_$rejdown1$	transition immédiate activée pour passer de l'état sleep vers l'état off
T_$rejdown2$	transition immédiate activée pour passer de l'état idle vers l'état off
T_$rejdown3$	transition immédiate activée pour passer de l'état fastactive vers l'état off
T_$rejdown4$	transition immédiate activée pour passer de l'état slowactive vers l'état off
T_$rejdown5$	transition immédiate activée pour passer de l'état wakeup vers l'état off
T_$notrejdown$	ttransition activée pour passer de l'état off vers l'état sleep

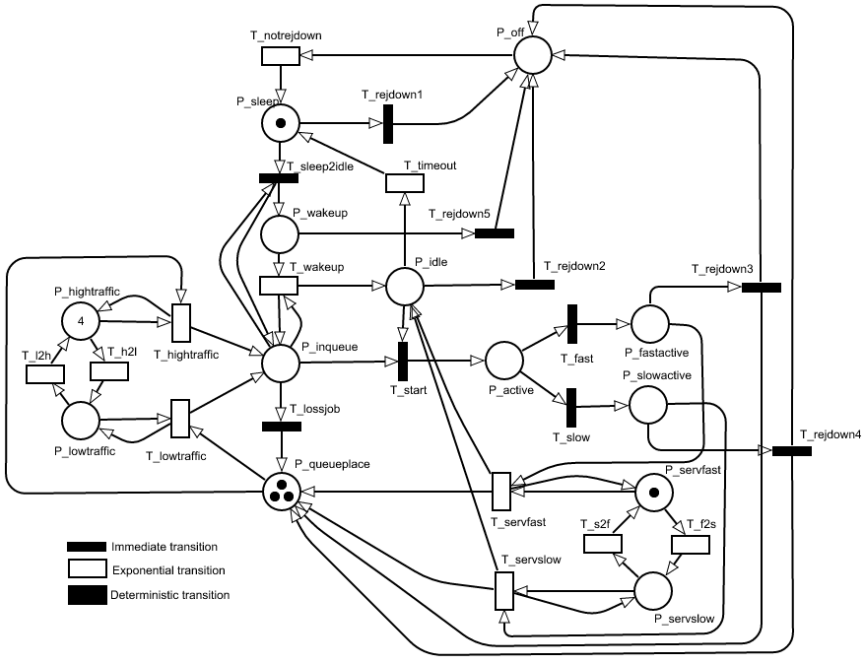


FIGURE 4.4 – Sous modèle workload-aware PM proposé

Description du processus de traitement d'une demande

Si on considère que le PMC est initialement dans l'état sleep (P_sleep), et aussitôt une requête arrive, la transition T_sleep2wakeup est activée. Ensuite, un jeton est déposé dans P_wakeup et la transition T_wakeup est ainsi activée. Son temps de franchissement (exponentiellement distribué) correspond au délai nécessaire pour le PMC pour transiter effectivement à l'état idle. Notons qu'à chaque fois que T_wakeup est activée et avant son franchissement, d'autres requêtes peuvent entrer au système (P_inqueue) et autant de jeton sont déposés dans P_inqueue. Dès que T_wakeup est franchie le PMC transite de l'état wakeup à l'état idle et un jeton est déposé dans P_idle. Ceci active aussitôt la transition immédiate T_start dans la mesure où au moins un jeton est présent dans la place P_inqueue (resp. P_idle). Dès que la transition immédiate T_start est franchie, un jeton est déplacé de chacune des places P_inqueue et P_idle et un autre jeton est mis dans P_active. A ce moment-là commence effectivement le traitement de la requête. Notons que le taux de traitement du service est modélisé avec un processus MMPP (Markov Modulated Poisson Process) doublement stochastiques à deux états. Ce processus permet de capturer la mise à l'échelle dynamique de la vitesse du PMC (exprimée par T_servfast et T_servslow) alternant entre P_servfast et P_servslow (en utilisant respectivement les transitions T_f2s et T_s2f). Selon les gardes assignées, une des transitions immédiates suivantes T_fast et T_slow est activée. T_fast (resp. T_slow), une fois franchie, dépose un jeton dans P_fastactive (resp. dans P_slowactive). Ces gardes indiquent si le taux de traitement du service du PMC peut passer d'une vitesse rapide à lente ou inversement. Afin d'adapter le taux de service à la charge, nous utilisons un taux de franchissement avec marquage dépendant. L'objectif étant de quantifier cette adaptation en assignant une garde à une des deux transitions suivantes T_s2f et T_f2s. Ces gardes capturent le niveau de la charge.

Ainsi, pour un trafic décroissant d'un haut niveau à un bas niveau (ceci est quantifié en testant le nombre de jetons dans P_hightraffic et/ou P_lowtraffic, par exemple on peut fixer $\neq$ P_hightraffic = 0 comme une garde assignée à la transition T_f2s afin d'exprimer la valeur seuil au delà de laquelle le service commute d'un taux rapide à un taux lent). Ceci est réalisé en franchissant la transition T_f2s qui déplace un jeton de la place P_servfast et met un jeton dans la place P_servslow. Inversement pour un trafic croissant au delà d'un certaine valeur (quantifiée de la même manière comme dans le cas précédent, on fixe par exemple le seuil à $\neq$ P_hightraffic > 1 comme garde assignée à T_s2f afin d'exprimer la valeur seuil au delà de laquelle le service commute du taux lent au

taux rapide) pour que l'état du service transite de slow à fast. Il est à mentionner que les seuils sont choisis afin de limiter les oscillations entre les taux de service fast et slow. La distribution MMPP pour les requêtes d'arrivée est utilisée pour capturer la nature sporadique du trafic. Aussitôt la requête entièrement traitée, en franchissant la transition T_servfast ou la transition T_servslow, le PMC quitte l'état idle et déclenche en même temps un temporisateur de durée T_timeout avec une distribution déterministe. Si une nouvelle requête arrive avant l'expiration du timeout alors la transition T_timeout est préemptée (avec la politique prd) et le PMC revient de l'état idle (P_idle) à l'état active (P_active) après franchissement de T_start. Ainsi, le traitement de la nouvelle requête commence immdéiatement. Sinon, si ce timeout expire et le délai correspondant est écoulé, sans détecter durant cet intervalle de temps aucune arrivée de requête, alors la transition T_timeout est franchie et le mécanisme PM force le PMC à transiter immédiatement vers l'état sleep (P_sleep).
Finalement comme notre modèle versatile SRN SVS a pour objectif l'analyse de la performabilité, il est évident qu'il doit prendre en considération l'occurrence des défaillances. Ainsi, si à un certain moment la VM ou le VMM devient indisponible en raison d'une défaillance ou d'un rajeunissement, le PMC commute de l'état courant à l'état d'énergie off (P_off). Ceci est comptabilisé dans le SRN par les transitions immédiates respectives T_rejdown1, T_rejdown2, T_rejdown3, T_rejdown4. Ces dernières sont activées avec la même garde spécifiant que la VM ou le VMM est devenu indisponible. Dans un tel cas, une et une seule des transitions précédemment citées est franchie et déplace un jeton de l'une des places suivantes : P_sleep, P_idle, P_fastactive, P_slowactive à la place P_off.
Dans ce qui suit, on suppose que le trafic est constitué par un ensemble de requêtes de nature sporadique. Nous supposons également que ce trafic alterne entre des périodes on et des périodes off. Au cours des périodes où survient une requête et étant donné que le PMC est initialement dans un état de sleep, l'initiation du service (c.à.d. le traitement de la requête) n'est pas immédiatement accordée. Au lieu de cela, elle est reportée et déclenchée après qu'un délai donné soit déjà écoulé. Ce délai correspond à un certain temps nécessaire pour le PMC pour passer de l'état de veille (sleep) à l'état idle.

Pour résumer le processus ci-dessus, on définit Tint l'intervalle de temps entre l'instant où la requête est entièrement traitée et l'instant où la requête suivante arrive. Dès que le traitement de la requête précédente est terminé, PMC passe immédiatement de l'état actif à l'état P_idle. Si Tint est strictement plus court qu' au délai T_timeout) alors PMC reste en P_idle jusqu'à l'arrivé du

TABLEAU 4.4 – Distributions de transitions et gardes assignées pour le sous workload-aware PM(Cold-VM rejuvenation)

Transitions	**gardes**
T_$servfast$	($P\_vup == 1$) ‖ ($P\_vfp == 1$) ‖ ($P\_vsd == 1$) ‖ ($P\_vfpsd == 1$)
T_$servslow$	($P\_vup == 1$) ‖ ($P\_vfp == 1$) ‖ ($P\_vsd == 1$) ‖ (_ == 1)
T_$f2s$	($P\_hightraffic == 0$) && ($P\_vup == 1$) ‖ ($P\_vfp == 1$) ‖ ($P\_vsd == 1$) ‖ ($P\_vfpsd == 1$)
T_$s2f$	($P\_hightraffic > 0$) && ($P\_vup == 1$) ‖ ($P\_vfp == 1$) ‖ ($P\_vsd == 1$) ‖ ($P\_vfpsd == 1$)
T_$fast$	($P\_servfast == 1$)
T_$slow$	($P\_servslow == 1$)
T_$start$	($P\_vup == 1$) ‖ ($P\_vfp == 1$) ‖ ($P\_vsd == 1$) ‖ ($P\_vfpsd == 1$)
$T\_sleep2wakeup$	($P\_vup == 1$) ‖ ($P\_vfp == 1$) ‖ ($P\_vsd == 1$) ‖ ($P\_vfpsd == 1$)
T_$wakeup$	($P\_vup == 1$) ‖ ($P\_vfp == 1$) ‖ ($P\_vsd == 1$) ‖ ($P\_vfpsd == 1$)
T_$timeout$	($P\_vup == 1$) ‖ ($P\_vfp == 1$) ‖ ($P\_vsd == 1$) ‖ ($P\_vfpsd == 1$)
T_$lossjob$	($P\_vup == 0$) && ($P\_vfp == 0$) && ($P\_vsd == 0$) && ($P\_vfpsd == 0$)
T_$rejdown1$	($P\_vup == 0$) && ($P\_vfp == 0$) && ($P\_vsd == 0$) && ($P\_vfpsd == 0$)
T_$rejdown2$	($P\_vup == 0$) && ($P\_vfp == 0$) && ($P\_vsd == 0$) && ($P\_vfpsd == 0$)
T_$rejdown3$	($P\_vup == 0$) && ($P\_vfp == 0$) && ($P\_vsd == 0$) && ($P\_vfpsd == 0$)
T_$rejdown4$	($P\_vup == 0$) && ($P\_vfp == 0$) && ($P\_vsd == 0$) && ($P\_vfpsd == 0$)
T_$rejdown5$	($P\_vup == 0$) && ($P\_vfp == 0$) && ($P\_vsd == 0$) && ($P\_vfpsd == 0$)
T_$notrejdown$	($P\_vup == 1$) ‖ ($P\_vfp == 1$) ‖ ($P\_vsd == 1$) ‖ ($P\_vfpsd == 1$)

nouvelle requête, et à cet instant il passe à l'état actif pour lancer le traitement de la demande en cours. Inversement si Tint est supérieur au délai T_timeout et dès que cette dernière est écoulée, PMC passe de l'état idle à l'état sleep. Ceci engendre un retard de traitement de la demande dû au temps nécessaire pour passer de l'état P_sleep à l'état P_idle.
Afin de bien comprendre comment les états PMC évoluent et comment les demandes sont traitées en fonction du temps, nous discuterons dans ce qui suit, les différents types de retard des transitions. Ceci permet d'expliquer le temps d'attente accumulé par une requête depuis son instant d'arrivée jusqu'à son départ du SVS.
Notez que selon notre approche de modélisation, le processus de traitement des demandes peut être considéré comme un système de mise en file d'attente particulier où le temps d'attente se compose de trois retards cumulatifs :

— Le premier, D1, est dû au délai de franchissement de la transition T_wakeup. Dans le meilleur des cas D1 = 0 (si l'arrivée de la demande se produit et trouve PMC en état idle) et dans le pire des cas D1 = délai de T_wakeup (si l'arrivée de la demande se produit et trouve le tampon vide et PMC en état sleep). Si une arrivée survient et que la transition Twakeup est déjà déclenchée par une arrivée précédente, alors D1 est égale au temps résiduel de Twakeup ; 0 <D1 <Twakeup.

— Le second, D2, est dû au temps d'attente dans la file d'attente SVS avant de démarrer le service ;

— Le troisième délai, D3, correspond au temps de traitement attendu de la demande (durée moyenne du service).

Les places invariantes (P-invariant)

Dans ce qui suit nous détaillons les places invariantes (P-invariant) du sous modèle workload-aware PM SRN (le symbole ≠ correspond au nombre des jetons dans une place) :

— Le premier P-invariant correspond à la propriété du système qu'il y ait au plus K places disponibles dans le buffer c.à.d. ≠P_queueplace + ≠P_inqueue + ≠P_active + ≠P_fastactive + ≠P_slowactive = K.

— Le second P-invariant correspond à la propriété que le PMC soit dans l'un des états possibles (i.e. ≠P_off + ≠P_sleep + ≠P_idle + ≠P_active + ≠P_slowactive + ≠P_fastactive = 1).

— Le troisième ≠P_servslow + ≠P_servfast = 1 décrit la propriété que le PMC, une fois actif, possède un taux fast ou slow. La propriété décrite par l'équation suivante ≠P_hightraffic +

TABLEAU 4.5 – Distributions de transitions et gardes assignées pour le sous workload-aware PM (Migrate-VM rejuvenation)

Transitions	**Gardes**
T_servfast	$(P\_vup == 1) \parallel (P\_vfp == 1) \parallel (P\_vmigd == 1) \parallel (P\_vfpmigd == 1)$
T_servslow	$(P\_vup == 1) \parallel (P\_vfp == 1) \parallel (P\_vmigd == 1) \parallel (P\_vfpmigd == 1)$
T_f2s	$(P\_hightraffic == 0)$ && $(P\_vup == 1) \parallel (P\_vfp == 1) \parallel (P\_vmigd == 1) \parallel (P\_vfpmigd == 1)$
T_s2f	$(P\_hightraffic > 0)$ && $(P\_vup == 1) \parallel (P\_vfp == 1) \parallel (P\_vmigd == 1) \parallel (P\_vfpmigd == 1)$
T_fast	$(P\_servfast == 1)$
T_slow	$(P\_servslow == 1)$
T_start	$(P\_vup == 1) \parallel (P\_vfp == 1) \parallel (P\_vmigd == 1) \parallel (P\_vfpmigd == 1)$
T_sleep2wakeup	$(P\_vup == 1) \parallel (P\_vfp == 1) \parallel (P\_vmigd == 1) \parallel (P\_vfpmigd == 1)$
T_wakeup	$(P\_vup == 1) \parallel (P\_vfp == 1) \parallel (P\_vmigd == 1) \parallel (P\_vfpmigd == 1)$
T_timeout	$(P\_vup == 1) \parallel (P\_vfp == 1) \parallel (P\_vmigd == 1) \parallel (P\_vfpmigd == 1)$
T_lossjob	$(P\_vup == 0)$ && $(P\_vfp == 0)$ && $(P\_vmigd == 0)$ && $(P\_vfpmigd == 0)$
T_rejdown1	$(P\_vup == 0)$ && $(P\_vfp == 0)$ && $(Pvmigd == 0)$ && $(P\_vfpmigd == 0)$
T_rejdown2	$(P\_vup == 0)$ && $(P\_vfp == 0)$ && $(P\_vmigd == 0)$ && $(P\_vfpmigd == 0)$
T_rejdown3	$(P\_vup == 0)$ && $(P\_vfp == 0)$ && $(P\_vmigd == 0)$ && $(P\_vfpmigd == 0)$
T_rejdown4	$(P\_vup == 0)$ && $(P\_vfp == 0)$ && $(P\_vmigd == 0)$ && $(P\_vfpmigd == 0)$
T_rejdown5	$(P\_vup == 0)$ && $(P\_vfp == 0)$ && $(Pvmigd == 0)$ && $(P\_vfpmigd == 0)$
T_notrejdown	$(P\_vup == 1) \parallel (P\_vmigd == 0)$ && $(P\_vfpmigd == 0)$

≠P_lowtraffic = n signifie que le trafic d'arrivée est composé de n sources.

Paramètres ajustables dans la politique workload-aware PM proposée

Dans le modèle SRN décrivant le mécanisme PM (Fig. 4.6.2.3), nous distinguons deux types de transitions temporisées.

— Le premier comprend les transitions (comme T_notrejdown et T_wakeup) qui possèdent des délais non ajustables. Ces derniers représentent en fait des caractéristiques inhérentes de la technologie PMC. T_notrejdown correspond au temps nécessaire au PMC pour passer de l'état off à l'état sleep, alors que T_wakeup est le temps nécessaire pour passer de l'état sleep à l'état idle.

— Le deuxième type de transitions possède une temporisation ajustable en fonction du contexte et en fonction de la charge de travail. L'ajustement du délai de ces transitions de manière opportuniste est un facteur clé dans la prise d'une décision appropriée d'une politique donnée en se basant sur les attributs de la charge de travail. Ceci est le cas pour la transition T_timeout ainsi que les transitions modélisant le traitement du service c.à.d T_servfast, T_servslow, T_s2f et T_f2s. Le choix du délai T_timeout est d'une importance primordiale pour aborder la question du compromis énergie-performance. Les attributs de la politique sont étroitement liés à la dynamique de la charge de travail. En effet, l'adoption d'une politique inappropriée dans un contexte donné concernant une charge de travail donnée équivaut à ne pas prendre la bonne décision au bon moment. Cela conduira inévitablement à fournir des résultats insatisfaisants ne respectant pas les objectifs prédéfinis de la politique adoptée. Le comportement oscillatoire peut être évité en choisissant une valeur de T_timeout supérieure à un seuil donné.

États de gestion de l'énergie : Advanced Configuration and Power Interface (ACPI)

Réduire la consommation énergétique (dynamique et statique) des systèmes informatiques est devenu une préoccupation majeure pour leurs concepteurs. Le problème de la gestion des ressources énergétiques a été examiné dans la littérature pour différents contextes, au niveau composant, au niveau OS de serveurs virtualisés et non virtualisés, mais aussi au niveau cluster, data centers et cloud. Plusieurs solutions ont été implémentés soit au niveau hardware ou logiciel.
Plusieurs algorithmes DPM (Dynamic Power Management) de gestion dynamique de l'énergie peuvent être implémentés au niveau hardware comme partie intégrante d'un circuit électronique (BIOS). Cependant, une implémentation hardware présente ne donne pas une bonne gestion d'alimentation car l'utilisateur n'a pas une visibilité sur le paramétrage de la gestion de l'énergie.
Afin de traiter ce problème, les chercheurs ont réalisé l'implémentation au niveau logiciel. En 1996,

TABLEAU 4.6 – Paramètres utilisés dans les deux sous-modèles de disponibilité SVS

Symbole	Description	Valeure	Temps moyen
λ_{FPv}	taux de vieillissement de la VM	0.005952381	1 week
λ_v	taux de défaillance de la VM après vieillissement	0.013888889	3 days
δ_v	taux de détéction de la défaillance du VM	12	5 mins
μ_v	taux de recouvrement de la défaillance de la VM	2	30 mins
β_v	taux de rajeunissement de la VM	60	1 min
r_v	Taux de redémarrage de la VM	120	30 secs
σ_v	Taux d'arrêt de la VM	120	30 secs
ω_v	taux de migration de la VM	3600	1 sec
τ_v	taux de déclenchement du rajeunissement de la VM	0.041666667	1 day
λ_{FPh}	taux de vieillissement du VMM	0.001388889	1 month
λ_h	taux de défaillance du VMM après vieillissement	0.005952381	1 week
σ_h	taux de détéction de la défaillance du VMM	12	5 mins
μ_h	taux de recouvrement de la défaillance du VMM	1	1 hour
β_h	taux de rajeunissement du VMM	30	2 mins
τ_h	taux de déclenchement du rajeunissement du VMM	0.005952381	1 week

Intel, Microsoft, et Toshiba ont publié la première version de la norme ACPI. C'est un standard qui définit des interfaces pour la gestion des ressources et de l'énergie indépendamment des plateformes à fin de gérer l'énergie et monitorer le hardware. ACPI offre la possibilité d'intégrer la DPM dans le contrôle de l'OS et requiert des OS compatibles-ACPI afin de prendre en charge le système et d'avoir le contrôle de tous les aspects correspondants de gestion de l'énergie et de la configuration.
Il est important de noter aussi que l'ACPI fournit une interface qui peut être utilisée par les développeurs de logiciels pour tirer parti de la souplesse dans l'ajustement des états d'énergie du système. Cette norme utilise les états basse-consommation du processeur. Les états les plus pertinents dans ce contexte sont C-states et P-states. Les C-states sont des états d'énergie (C0-C3) pour le CPU appelés respectivement Operating State, Halt, Stop-Clock, and Sleep Mode. En outre, lorsque le processeur est opérationnel, il peut être dans l'un des états d'énergie power-performance (P-states). Chacun de ces états désigne une combinaison particulière des configurations DVFS. Les P-states dépendent de l'implémentation, mais P0 est toujours l'état ayant la plus haute performance.

TABLEAU 4.7 – Valeurs des transitions temporisées utilisées dans le sous-modèle workload-aware PM SRN

transition	**Value**
T$_h2l = \frac{1}{\sigma_{T_h2l}}$	0.1
T$_l2h = \frac{1}{\sigma_{T_l2h}}$	0.01
T$_hightraffic = \frac{1}{\lambda_{T_hightraffic}}$	[0.0015,...,250]
T$_lowtraffic = \frac{1}{\lambda_{T_lowtraffic}}$	[0.0003,...,50]
T$s_ervfast = \frac{1}{\lambda_{T_servfast}}$	0.05
T$_servslow = \frac{1}{\lambda_{T_servslow}}$	0.002
T$_f2s = \frac{1}{\sigma_{T_f2s}}$	0.1
T$_s2f = \frac{1}{\sigma_{T_s2f}}$	0.01
T$_wakeup = \frac{1}{\lambda_{T_wakeup}}$	0.01
T$_timeout = \frac{1}{\lambda_{T_timeout}}$	0.02

4.7 Paramètres et métriques du modéle SVS

Dans cette section nous définissons à partir du modèle versatile SVS certains paramètres et métriques SVS sélectionnés (paramètres de trafic, facteur de vieillissement, métriques de performabilité, métriques de puissance et de performance de puissance) qui seront exploitées lors de l'analyse numérique dans la section suivante.

4.7.1 Paramètres de trafic

Comme nous étudions l'impact de la charge de travail sur les paramètres de performabilité, nous définissons dans cette sous section les paramètres de trafic. Un processus MMPP modélise le trafic qui est de nature variable et sporadique. C'est un processus de Poisson doublement stochastique à deux états. Le trafic global est généré par la superposition de plusieurs processus ponctuels. Le processus MMPP à 2 états est caractérisé par quatre paramètres : deux taux de transition permettant de passer d'un débit vers un autre (σ_{T_h2l} : taux de transition du débit élevé au débit bas, σ_{T_l2h} : taux de transition d'un débit bas au débit élevé) et deux taux d'arrivée ($\lambda_{T\_hightraffic}$ et $\lambda_{T\_lowtraffic}$). **Taux moyen des arrivées (λ)** : est le nombre moyen des requêtes arrivant à SVS est donné par :

$$\lambda = \frac{\lambda_{T\_hightraffic}.\sigma_{T\_l2h} + \lambda_{T\_lowtraffic}.\sigma_{T\_h2l}}{\sigma_{T\_h2l} + \sigma_{T\_l2h}} \tag{4.1}$$

Indice de dispersion (Index of dispersion for counts (IDC) : c'est un indicateur des rafales dans le trafic arrivant à SVS. La formule de l'IDC d'un processus MMPP à 2 états a été dérivée par Heffes et Lucantoni dans [117]. Ainsi, l'Indice de dispersion (IDC) est donné par :

$$\text{IDC} = 1 + \frac{2(\lambda_{T\_hightraffic} - \lambda_{T\_lowtraffic})^2 \sigma_{T\_h2l}.\sigma_{T\_l2h}}{(\sigma_{T\_h2l} + \sigma_{T\_l2h})^2 (\lambda_{T\_hightraffic}.\sigma_{T\_l2h} + \lambda_{T\_lowtraffic}.\sigma_{T\_h2l})} \tag{4.2}$$

4.7.2 Facteur de vieillissement du SVS (AF)

Plusieurs études ont montré la corrélation entre la variation de la charge de travail et le vieillissement des logiciels. En d'autres termes, le vieillissement des logiciels est dépendant à la charge de travail. Nous définissons le facteur de vieillissement comme une pondération du taux de vieillissement pour VM et VMM. Une telle définition est utilisée afin d'étudier l'impact de la fluctuation de la charge de travail sur les métriques de performance du SVS. Notez que dans cette thèse, nous étudions le cas où le vieillissement logiciel dépend à la fois du temps et de la charge de travail.Le facteur de vieillissement sera compté dans la transition T_hpf.

Définition : Soit AF (Eq. 4.3) le facteur de vieillissement dépendant de la charge de travail défini comme le rapport entre la somme pondérée des puissances de tous les états PMC (sleep, , wakeup, idle, slowactive, fastactive) et la puissance correspondant à PMC à l'état idle. Nous choisissons ce dernier comme état de puissance de référence. Notez que la somme pondérée des puissances citées ci-dessus englobe la somme de chaque valeur de puissance d'un état PMC donné multiplié par la probabilité que le PMC soit dans cet état.

$$\begin{aligned} \text{AF} = \text{E}\{\#\text{P}\_idle\} + \frac{Powersleep}{Poweridle}.\text{E}\{\#\text{P}\_sleep\} + \frac{Powerwakeup}{Poweridle}.\text{E}\{\#\text{P}\_wakeup\} + \\ \frac{Powerslowactive}{Poweridle}.\text{E}\{\#\text{P}\_slowactive\} + \frac{Powerfastactive}{Poweridle}.\text{E}\{\#\text{P}\_fastactive\} \end{aligned} \tag{4.3}$$

Selon le choix que nous avons fait, AF peut être plus petit, égal ou supérieur à 1. Si

1. AF <1, AF permet de ralentir le vieillissement et donc d'améliorer la disponibilité.
2. AF = 1, le vieillissement et la disponibilité du SVS sont insensibles au facteur de vieillissement. Par conséquent, le taux de vieillissement est indépendant de la charge de travail.
3. AF> 1, le vieillissement est accéléré et ainsi la disponibilité diminue avec l'augmentation de la charge de travail.

4.7.3 Métriques de performabilité

— **La disponibilité en régime permanent (Steady-state availability)** (As)

(Cold-VM) $$A_s = E\{\#P\_vup\} + E\{\#P\_vfp\} + E\{\#P\_vsd\} + E\{\#P\_vfpsd\} \quad (4.4)$$

(Migrate-VM) $$A_s = E\{\#P\_vup\} + E\{\#P\_vfp\} + E\{\#P\_vmigd\} + E\{\#P\_vfpmigd\} \quad (4.5)$$

— **le temps moyen d'attente :**En utilisant la loi de Little, le temps moyen d'attente accumulé par une requête dans le SVS peut être déduit à partir du modèle comme suit :

$$W = N/Th \quad (4.6)$$

— **le nombre moyen de requêtes** (N) dans le système, donné par :

$$N = E\{\#P\_inqueue\} + E\{\#P\_slowactive\} + E\{\#P\_fastactive\} \quad (4.7)$$

— **le throughput** (Th)**du système (en requêtes/s)**

$$Th = E\{\#T\_servfast\} + E\{\#T\_servslow\} \quad (4.8)$$

— **Fréquence de franchissement de T_lossjob (requêtes/seconde)** : quantifie le taux moyen de perte de requêtes dans SVS.

$$F\_Tlossjob = (E\{\#Phightraffic\}.\lambda_{Thightraffic} + E\{\#Plowtraffic\}.\lambda_{Tlowtraffic}).(E\{\#Pvstop\} + E\{\#Pvpre\} + E\{\#Pvfail\}) \quad (4.9)$$

— **Fréquence de franchissement de** T$_wakeup$ (F_T$wakeup$)

Cold-VM :

$$F\_Twakeup = \frac{A}{B}.\lambda_{T\_wakeup} \quad (4.10)$$

Migrate-VM :

$$F\_Twakeup = \frac{C}{B}.\lambda_{T\_wakeup} \quad (4.11)$$

où

$$A = (E\{\#P\_vup\} + E\{\#P\_vfp\} + E\{\#P\_vsd\} + E\{\#P\_vpsd\}) + E\{\#P\_wakeup\}.E\{\#P\_inqueue\} \quad (4.12)$$

$$B = E\{\#P\_inqueue\} + E\{\#P\_queueplace\} + E\{\#P\_active\} + E\{\#P\_slowactive\} + E\{\#P\_fastactive\} \quad (4.13)$$

$$C = (E\{\#P\_vup\} + E\{\#P\_vfp\} + E\{\#P\_vmigd\} + E\{\#P\_vfmigd\}) + E\{\#P\_wakeup\}.E\{\#P\_inqueue\} \quad (4.14)$$

4.7.4 Métriques de puissance

— **Puissance moyenne consommée par PMC** (P) **(watt)**

$$P = Powersleep.E\{\#P\_sleep\} + Poweridle.E\{\#P\_idle\} + Powerwakeup.E\{\#P\_wakeup\} + Powerslowactive.E\{\#P\_slowactive\} + Powerfastactive.E\{\#P\_fastactive\} \quad (4.15)$$

— **taux d'utilisation de la puissance** (U) : est défini comme le rapport entre FoTP$used$ et FoTPreq :

$$\mathrm{U} = \mathrm{F}o\mathrm{TP}used/\mathrm{F}o\mathrm{TP}req \tag{4.16}$$

où FoTP$used$ est la fraction de temps pendant laquelle le PMC est dans un état actif :

$$\mathrm{F}o\mathrm{TP}used = \mathrm{E}\{1_{\#\mathrm{P}\_fastactive=1}\} + \mathrm{E}\{1_{\#\mathrm{P}\_slowactive=1}\} \tag{4.17}$$

et FoTPreq est la fraction de temps le PMC est dans un état consommant de l'énergie (c.à.d. slowactive, fastactive, wakeup, idle, sleep) est donné par :

$$\mathrm{F}o\mathrm{TP}req = \mathrm{E}\{1_{\#\mathrm{P}\_fastactive=1}\} + \mathrm{E}\{1_{\#\mathrm{P}\_slowactive=1}\} + \mathrm{E}\{1_{\#\mathrm{P}\_wakeup=1}\} + \mathrm{E}\{1_{\#\mathrm{P}\_idle=1}\} + \mathrm{E}\{1_{\#\mathrm{P}\_sleep=1}\} = \mathrm{E}\{1_{\#\mathrm{P}\_off=0}\} \tag{4.18}$$

4.7.5 Métriques de puissance-performance

— **Efficacité** (E) **(joules /requête)** : est le rapport entre la puissance moyenne consomméer P (Eq. 4.15) et le throughput (du système Th (Eq. 4.8)

$$\mathrm{E} = \frac{\mathrm{P}}{\mathrm{T}h} \tag{4.19}$$

4.8 Résultats numériques

Dans cette section nous investiguons quelques résultats numériques relatifs au modèle SRN SVS pour les deux types de rejuvenation (model[1] et model[2]). Nous nous focalisons en particulier sur l'impact de la variation de la charge de travail sur la dynamique et le comportement du SVS. Pour mettre en évidence l'influence du charge de travail sur les paramètres et les métriques SVS définis dans la section précédente, nous avons effectué une série d'expériences basées sur l'analyse analytique en utilisant l'outil SPNica tool. Dans ce qui suit, nous étudions la performabilité de l'analyse numérique et les mesures de puissance de SVS.
Le traitement de trafic en rafale permet à servir dans le même cycle un lot de requêtes. Ceci est susceptible d'être réalisé par PMC avec moins de temps en état idle et plus de temps en état de Sleep. Étant donné que l'état idle consomme beaucoup plus d'énergie que l'état de Sleep, le traitement par lots permet une meilleure performabilité.

4.8.1 Impact de la charge de travail sur le facteur de vieillissement (AF)

La figure 4.5 montre l'évolution du facteur de vieillissement AF en fonction de la charge de travail (λ). De toute évidence, le facteur de vieillissement est une fonction croissante de la charge de travail. Selon le graphique (Fig. 4.5), on distingue trois phases bien différenciées. Le premier correspond à un faible trafic où l'AF augmente lentement. Dans le second, l'AF augmente fortement par rapport à l'évolution de la charge de travail. Dans la dernière phase, correspondant à un SVS surchargé (au-delà d'une certaine charge), le facteur de vieillissement a tendance à converger vers une valeur de saturation où il est quasiment insensible aux variations de charge de travail et où la consommation d'énergie PMC est maximale.

4.8.2 Impact de la charge de travail sur les métriques des modèles SVS SRN polyvalents

4.8.2.1 Impact de la charge de travail sur les métriques de performance des modèles SVS SRN polyvalents

Disponibilité SVS (As) (Eq. 4.4 and Eq. 4.5)
Nous observons à partir des courbes de la Fig. 4.6 que l'impact de la charge de travail sur la disponibilité en régime permanent (As) est clairement le plus significatif pour une charge de travail

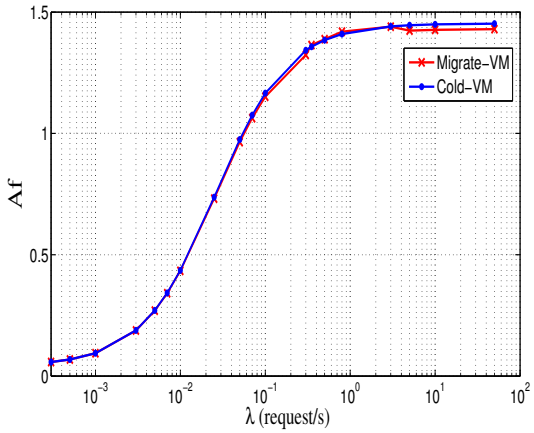


FIGURE 4.5 – facteur de vieillissement (AF) en fonction du λ

comprise entre 0,001 et 1. En dehors de cette intervalle, As est presque insensible aux variations de charge de travail. En effet, dans cette intervalle, comme on a constaté dans la figure 4.5 la charge de travail n'a pas d'impact significatif sur le vieillissement du logiciel et par conséquent n'a pas d'impact sur la disponibilité, alors que pour une charge de travail élevée, le taux de vieillissement est à sa valeur la plus élevée et une augmentation de la charge de travail n'affectera pas davantage la disponibilité. Notez également que pour les deux techniques de rajeunissement (Cold-Migrate et Migrate-VM) As suit le rythme de l'AF. Ce comportement est relativement simple dans la mesure où la disponibilité évolue proportionnellement au facteur de vieillissement.
Il convient également de mentionner que model^2 présente une meilleure disponibilité que model^1

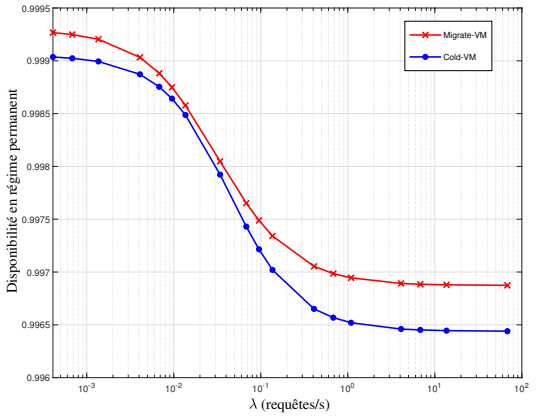


FIGURE 4.6 – Disponibilité en régime permanent (As) en fonction de λ

pour une charge de travail donnée. Ce résultat est prévisible car l'indisponibilité du SVS correspondant au temps passé pendant le processus de migration de VM (pour Migrate-VM) est nettement inférieur au temps nécessaire pour le rajeunissement de VMM (pour le rajeunissement de Cold-VM).

Temps moyen d'attente (*W*) (Eq. 4.6) :
La figure 4.7 montre le temps moyen d'attente de la requête par seconde en fonction de la charge de travail. Cela montre qu'à mesure que la charge de travail augmente, W augmente dans les deux modèles. Cela est évident car une charge plus importante dans SVS donne lieu à une file d'attente plus longue et donc plus de retard pour le traitement des demandes.

Fréquence moyenne de franchissement du T_$lossjob$ (F_T$lossjob$)**(Eq. 4.9) :** La figure 4.8 met en évidence l'évolution de F_T$lossjob$ en fonction de la charge de travail. On observe que

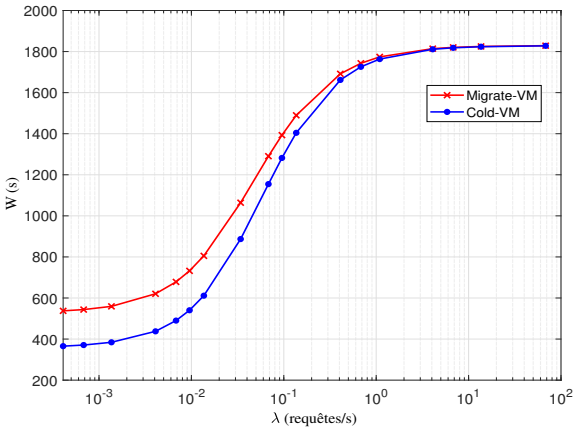


FIGURE 4.7 – Temps moyen d'attente en fonction de λ

F_T$lossjob$ augmente en fonction du taux moyen d'arrivée. Ceci est également simple car plus la charge est élevée dans SVS, plus les pertes de demandes sont importantes (puisque la file d'attente a une capacité finie). Notant également que model[1] subit plus de pertes que model[2] pour n'importe quelle charge de travail. Cela peut être justifié par les mêmes arguments donnés ci-dessus pour le comportement de la disponibilité en régime permanent du SVS (Fig. 4.6) par rapport à la charge de travail.

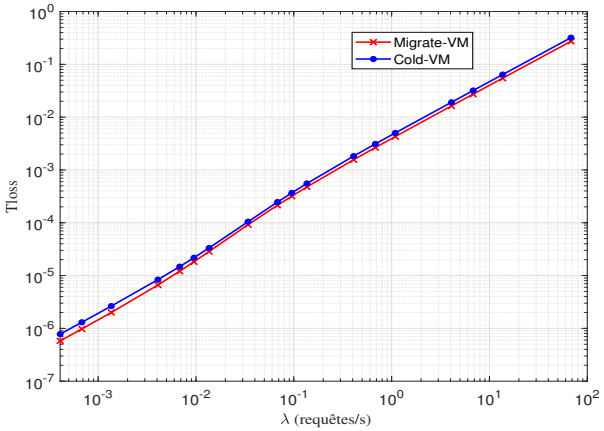


FIGURE 4.8 – Fréquence moyenne de franchissement du T_$lossjob$ en fonction de λ

Fréquence moyenne de franchissement du T_$wakeup$ **(**F_T$wakeup$**) (Eq. 4.11, Eq. 4.10)** Nous étudions ici l'évolution de la fréquence moyenne de franchissement de T_$wakeup$ (F_T$wakeup$). Nous observons dans la figure 4.9 que F_T$wakeup$ est une fonction croissante de la charge de travail. Plusieurs paramètres peuvent être impliqués dans l'explication d'un tel comportement. Le délai T_$wakeup$ ($\frac{1}{\lambda_{T wakeup}}$, Tab.4.7) représente le temps nécessaire pour passer de l'état Sleep à l'état idle.

Selon la Fig. 4.4, trois conditions doivent être remplies simultanément pour activer T_$wakeup$: *(i)* VM est disponible (voir la garde T_$wakeup$ dans Tab.4.4, Tab. 4.5), *(ii)* il y a au moins un jeton dans P_$inqueue$, *(iii)* il y a un jeton dans P_$sleep$ (résultant du franchissement de T_$notrejdown$ ou T_$timeout$). Ainsi, le franchissement de T_$wakeup$ est étroitement corrélé au déclenchement d'un ensemble de transitions à savoir T_$notrejdown$, T_$timeout$, T_$hightraffic$ et T_$lowtraffic$. L'évolution des états d'énergie du PMC et la manière de traitement des requêtes dans le SVS en fonction du temps expliquent l'évolution de la fréquence de franchissement de F_T$wakeup$. On définit un cycle comme étant la période de temps entre l'instant de déclenchement de T_$wakeup$

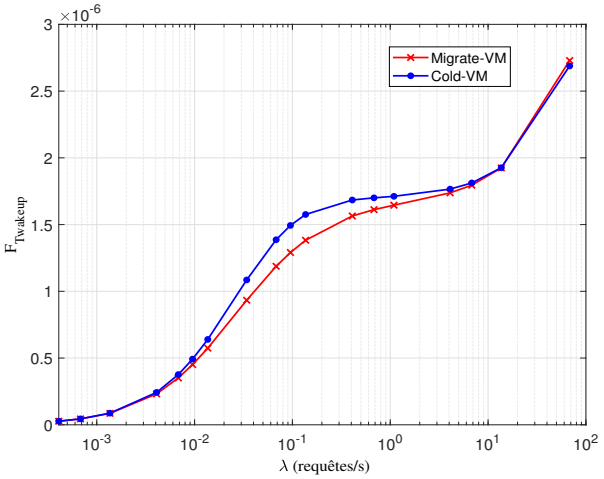


FIGURE 4.9 – Fréquence moyenne de franchissement du T_$wakeup$ F_T$wakeup$) en fonction de λ

et l'expiration de T_$timeout$. Pour chaque cycle correspond une seule occurrence de tir de T_$wakeup$ et de T_$timeout$. On considére trois types de charges de travail supportées par le SVS.

— Pour un trafic très léger (λ <10^{-2}), l'augmentation de la charge de travail n'affecte pas de manière significative la fréquence de franchissement de T_$hightraffic$ ou T_$lowtraffic$ (qui reste très faible) ce qui maintient P_$inqueue$ presque vide. Dans ce cas, même s'il existe souvent un jeton dans P_$wakeup$, T_$wakeup$ se déclenche très rarement.

— Pour un trafic moyen (λ ∈ [10^{-2}, 10^2]), l'augmentation de la charge de travail augmente la fréquence de commutation de PMC entre les différents états ACPI (sleep, wakeup idle, fastactive, slowactive et off). Il augmente également la fréquence de basculement de SVS entre les états de disponibilité et d'indisponibilité (ce qui augmente T_$notrejdown$). Tous ces événements expliquent l'augmentation de la fréquence F_T$wakeup$.

— Pour un trafic intense (λ > 10^4), l'augmentation de la charge de travail augmente le nombre de jetons dans P_$inqueue$ (s'il n'est pas plein) et par conséquent, la fréquence de franchissement de T_$wakeup$ dépend presque uniquement de la présence de jetons dans P_$wakeup$. Cette dernière condition équivaut à avoir un jeton en P_$sleep$ qui dépend à son tour du franchissement de T_$notrejdown$ ou T_T$imeout$).

4.8.2.2 Impact de la charge de travail sur les états d'alimentation ACPI du PMC

Dans cette partie, nous nous intéressons à l'évolution des états ACPI en fonction de la charge. Cette étude nous permet de mieux comprendre le comportement de SVS en fonction de la charge de travail. La figure 4.10 montre que la probabilité que le PMC soit en état wakeup (#P_$wakeup$) en fonction de la charge de travail est une fonction croissante. Notez que la dynamique des courbes suit la tendance du facteur de vieillissement par rapport à la charge de travail.

La figure 4.11 représente la probabilité du PMC à être dans l'état idle (*#P_idle*) en fonction de la charge de travail. Pour un trafic léger, la courbe de (*#P_idle*) évolue avec une pente positive jusqu'à atteindre un maximum au-delà duquel la courbe commence à décliner. En effet, pour un trafic relativement faible, le temps entre les arrivées des requêtes est beaucoup plus grand que le délai de Timeout. Ainsi, chaque traitement d'une requête est susceptible d'être suivi d'une période de temporisation complète.

Par conséquent, pour un trafic léger, à mesure que la charge de travail augmente, le temps moyen entre les requêtes diminue, ce qui entraîne plus de demandes et des périodes d'inactivité (idle) plus fréquentes. Au-delà d'un seuil de charge de travail donné (3.10^{-2}), le temps d'arrivées entre des requêtes tombe est inférieur à la valeur du délai de Timeout et dans ce cas, la demande suivante sera probablement traitée avant l'expiration du délai.

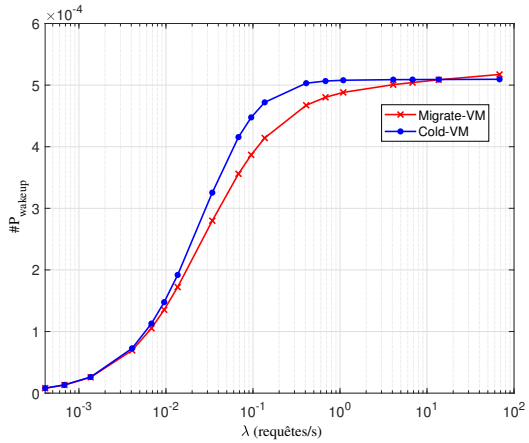


FIGURE 4.10 – #P_wakeup en fonction de λ

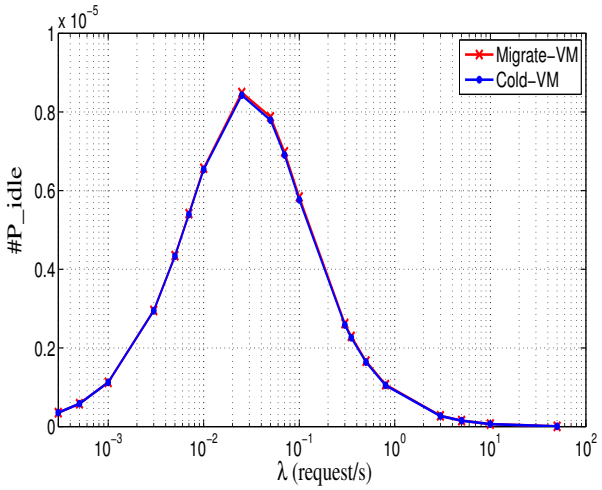


FIGURE 4.11 – #P_idle en fonction de λ

Pour un trafic intense, le temps entre les demandes devient beaucoup plus petit que le délai Timeout et, par conséquent, le PMC traiterait un lot de demandes avant de connaître un délai de Timeout complet. Ainsi, pour un trafic léger, chaque traitement de requête correspond en moyenne à une période de Timeout, alors que ce dernier est susceptible de correspondre à une taille de lot supérieure à un pour un trafic intense.
De toute évidence, à mesure que la charge de travail augmente, la probabilité de PMC d'être en état Sleep diminue (figure 4.12). En effet, l'augmentation de la charge de travail prolonge les périodes où PMC est active, ce qui raccourcit les périodes pendant lesquelles PMC est en état Sleep.

Les deux figure 4.14 et figure 4.13 illustrent comment les états de gestion de l'énergie (ACPI) de PMC évoluent pour différentes charges de travail. En fait, pour un trafic léger, l'état Sleep est l'état dominant et pour le trafic intense, les états de PMC les plus visités sont les états actifs (slowactive et fastactive).

4.8.3 Impact de la charge de travail sur les mesures de puissance

Puissance moyenne consommée (*P*) (Eq. 4.15)
La consommation d'énergie de PMC en fonction du taux moyen d'arrivée est représentée sur la figure 4.15. Pour les petits λ ($\lambda < 10^{-3}$), P est faible car PMC est presque toujours en état Sleep et donc consomme une petite valeur de puissance. Au fur et à mesure que λ augmente, PMC a

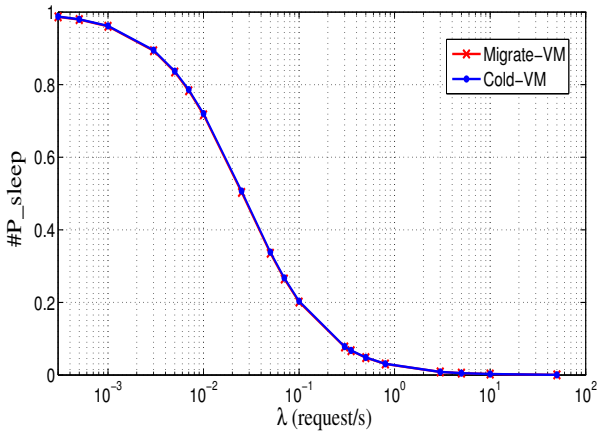


FIGURE 4.12 – #P_sleep en fonction de λ

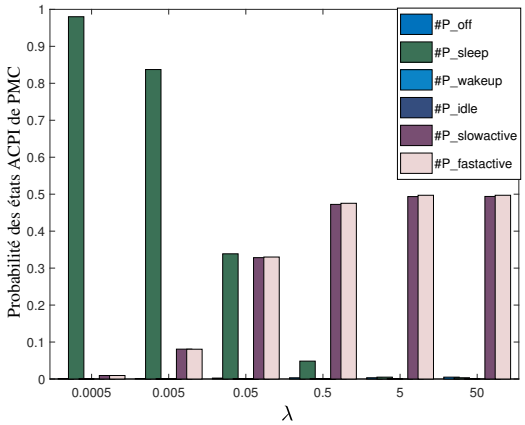


FIGURE 4.13 – Les états de gestion de l'énergie (ACPI) de PMC (Cold-VM)

tendance à aller plus vers des états actifs et ainsi P augmente et tend à converger vers une valeur fixe (cela correspond au seuil de puissance maximale où PMC est presque toujours à l'état actif). L'explication ci-dessus est valable à la fois pour model[1] et model[2].

Taux d'utilisation de la puissance (*U*) (Eq. 4.16)

La figure 4.16 représente U en fonction de λ. La courbe est une fonction croissante. Pour une charge de travail légère, PMC reste pendant de longues périodes en état de veille (Sleep) ce qui explique la faible valeur du taux d'utilisation de la puissance. Avec l'augmentation de la charge de travail, PMC occupe la plupart du temps les états actifs et, par conséquent, U augmente jusqu'à convergence vers l'unité (où la file d'attente est pleine).

4.8.3.1 Impact de la charge de travail sur l'efficacité puissance-performance des modèles SVS

Efficacité (*E*)(Eq. 4.19)

La figure 4.17 montre l'évolution de l'efficacité E (l'énergie consommée en joules par requête) en fonction de λ. E diminue par rapport à la charge de travail. À mesure que la charge de travail augmente, la période de veille se réduit et la taille du lot de requêtes par cycle augmente (pour un délai d'expiration (Timeout) fixe.Conséquemment, l'énergie consommée par requête diminue en fonction de la charge de travail. Pour un trafic très léger (SVS très peu chargé), l'efficacité est relativement élevée car elle comprend l'énergie consommée pendant un cycle et une période de sommeil (Sleep). Pour le trafic intense (SVS chargé), la période de sommeil devient très courte et l'énergie consommée correspondante par requête devient négligeable. Par conséquent, l'énergie

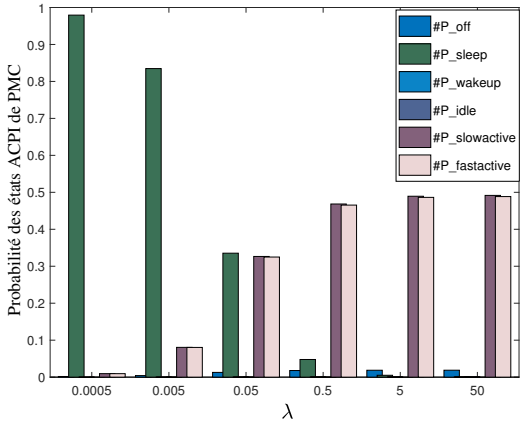


FIGURE 4.14 – Les états de gestion de l'énergie (ACPI) de PMC (Migrate-VM)

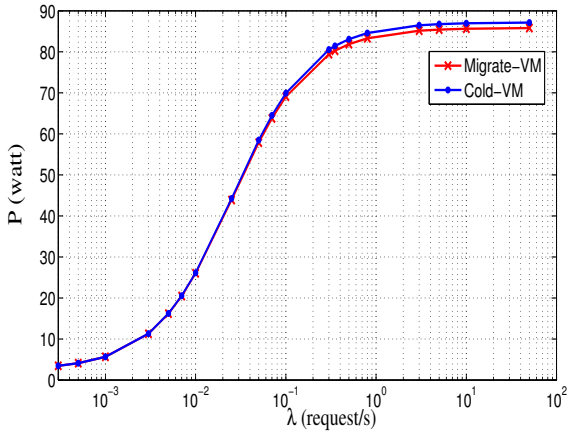


FIGURE 4.15 – Puissance moyenne consommée (P) en fonction de λ

consommée par requête est presque égale à l'énergie consommée pendant un cycle divisée par la taille du lot. Cela est évident car il est plus économique de traiter un lot de requêtes par cycle que de traiter chaque demande séparément.1

4.8.4 Impact de IDC sur les métriques des modèles SVS SRN polyvalents

À mesure que le trafic augmente, l'IDC augmente, ce qui produit un trafic en rafale. La figure 4.18 illustre comment les requêtes sont traitées dans le SVS en fonction du temps. L'augmentation de l'IDC influence d'une part l'évolution des états PMC et le traitement des requêtes en fonction du temps (pour différents IDC et constante λ) dans le SVS. En effet, le traitement d'un trafic plus en rafale équivaut à servir dans le même cycle un lot de grande taille de requêtes. Ainsi, le PMC passe moins de temps en état d'inactivité (idle) et plus de temps en état de veille (Sleep).

La figure 4.19 montre, pour un λ donné (correspondant à AF> 1), que la disponibilité en régime permanent du SVS est une fonction croissante de l'IDC et que le modèle[2] offre une meilleure disponibilité que le modèle[1]. Cette augmentation commence lentement au fur et à mesure que l'IDC augmente jusqu'à atteindre une certaine valeur IDC au-delà de laquelle elle devient beaucoup plus nette, en particulier pour un SVS chargé (λ = 10).

Pour un taux d'arrivée donné et à mesure que l'IDC augmente le temps moyen d'attente (W) (Fig.4.20), le taux moyen de perte (Fig.4.21) et la fréquence moyenne de franchissement de T_wakeup (Fig.4.22) diminuent. Cette diminution commence lentement à mesure que l'IDC augmente jusqu'à atteindre une certaine valeur IDC au-delà de laquelle elle devient plus aiguë, en particulier

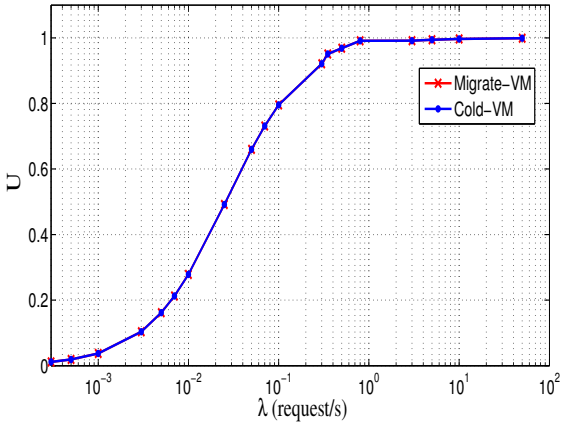


FIGURE 4.16 – Taux d'utilisation de la puissance (U) en fonction de λ

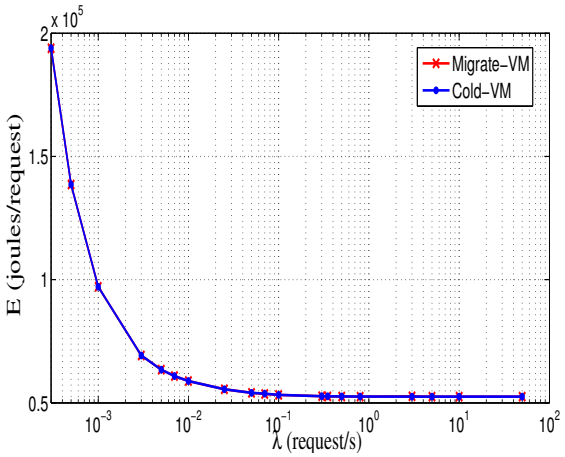


FIGURE 4.17 – Efficacité (E) en fonction de λ

pour le SVS chargé (λ = 10) où la taille du lot devient plus grande et la période de sommeil devient plus grande. Cela explique la forme de la courbe pour le trafic.

Pour un taux d'arrivée donné et au fur et à mesure que l'IDC augmente, *#P_wakeup* (Fig.4.23) diminue. Cette diminution commence lentement et au-delà d'un seuil donné devient beaucoup plus prononcée en particulier pour une charge de travail élevée (λ = 1, 10). L'étude du #P_idle (Fig.4.25) par rapport à l'IDC montre deux phases, une phase croissante et / ou une phase décroissante. Les mêmes arguments détaillés pour la **figure 12** peuvent être utilisés ici pour expliquer ce comportement. En effet, à mesure que IDC augmente, le trafic devient plus en raffale et les requêtes sont très susceptibles d'être traitées en lot. Cela signifie que pour une charge de travail donnée, PMC reste moins en état de repos et plus en état de veille (Fig. 4.26).

Pour les mêmes raisons citées ci-dessus dans cette sous-section et au-delà d'une valeur IDC donnée, les métriques suivantes, la puissance moyenne consommée par PMC (P) (Fig.26), le taux d'utilisation de la puissance (U) (Fig.27), l'efficacité (E) (Fig 28), diminuent fortement par rapport à IDC.

En conclusion, on peut en déduire que pour un taux moyen de demandes fixe (charge de travail) et pour un facteur de vieillissement AF> 1, l'état de rafale permet d'assembler les demandes sous forme de lots. Au fur et à mesure que l'IDC augmente, la taille des lots augmente, ce qui améliore considérablement tous les attributs SVS de performabilité étudiés.

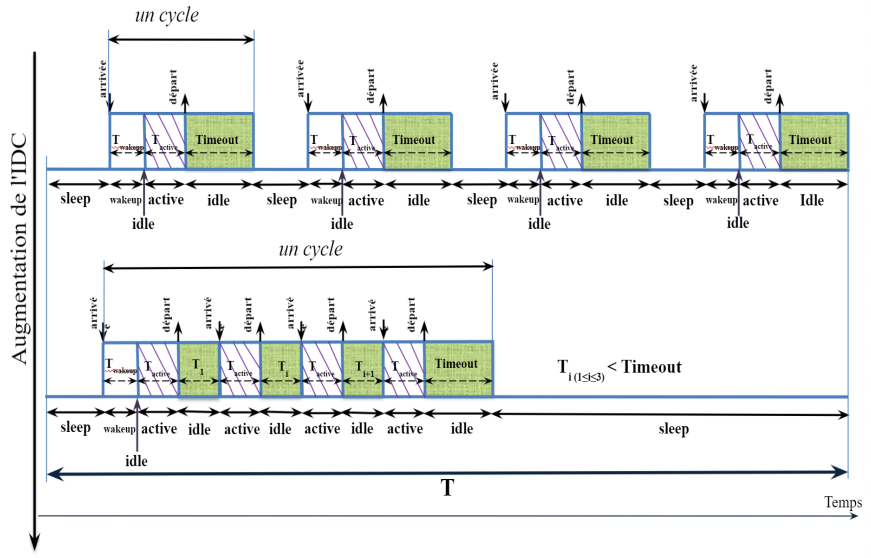


FIGURE 4.18 – Illustration de l'évolution des états d'énergie du PMC et traitement des requêtes en fonction du temps dans le SVS (pour différents IDC) dans le SVS

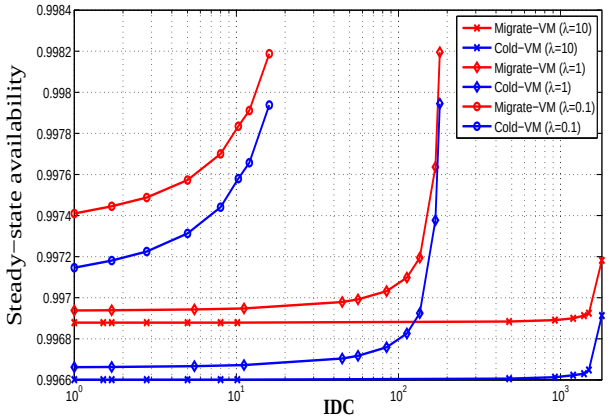


FIGURE 4.19 – Disponibilité en régime permanent (A*s*) en fonction de IDC (λ = 0.1, 1, 10)

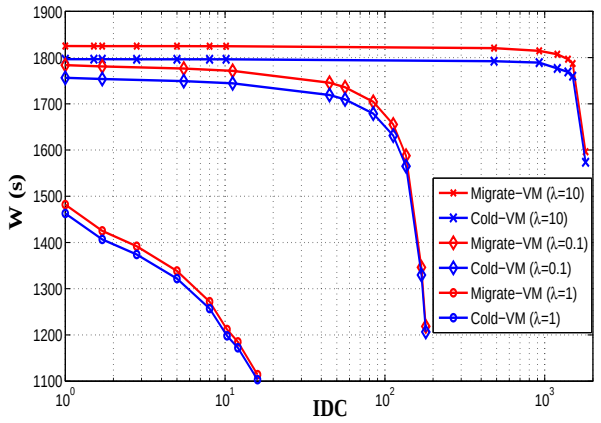


FIGURE 4.20 – Temps moyen d'attente en fonction de IDC (λ = 0.1, 1, 10)

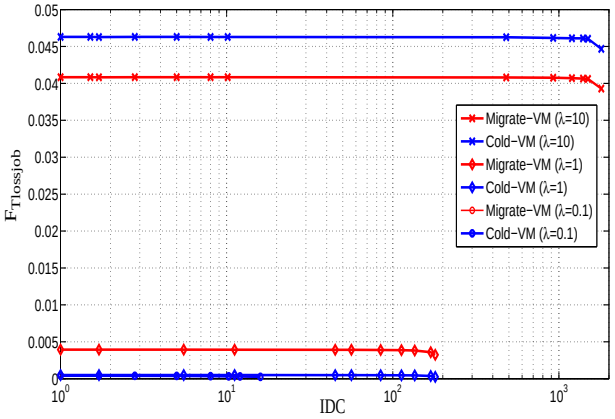


FIGURE 4.21 – F_T$lossjob$ en fonction de IDC (λ = 0.1, 1, 10)

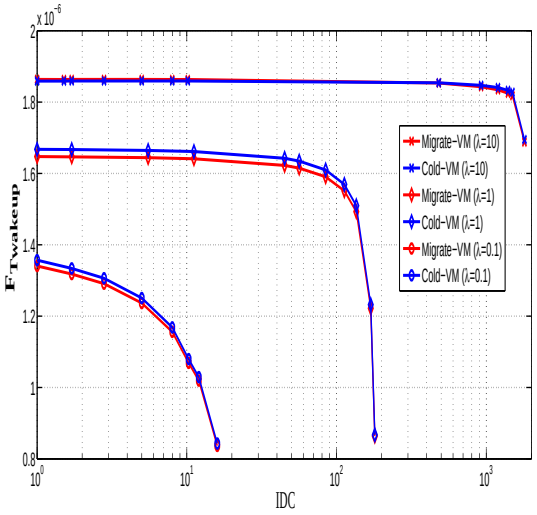


FIGURE 4.22 – F_T$wakeup$ en fonction de IDC (λ = 0.1, 1, 10)

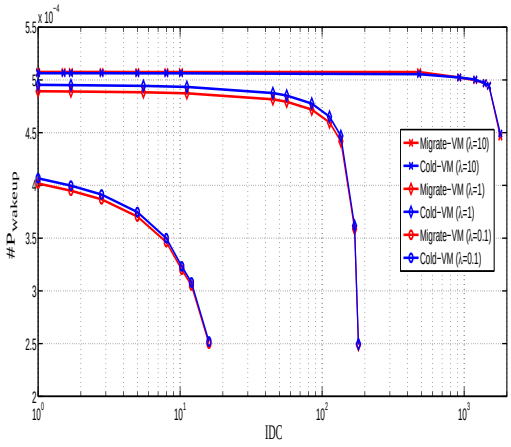


FIGURE 4.23 – *#P_wakeup* en fonction de IDC (λ = 0.1, 1, 10)

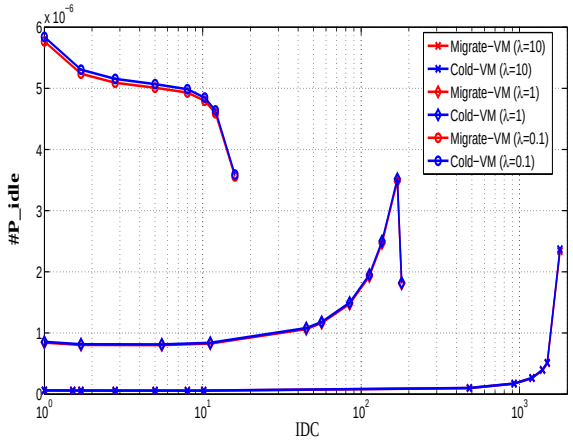


FIGURE 4.24 – *#P_idle* en fonction de IDC (λ = 0.1, 1, 10)

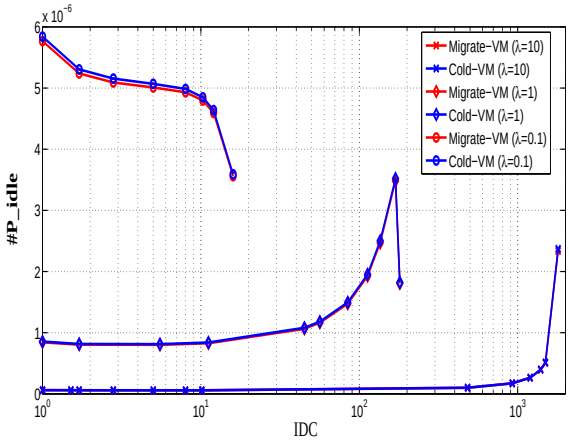


FIGURE 4.25 – *#P_idle* en fonction de IDC (λ = 0.1, 1, 10)

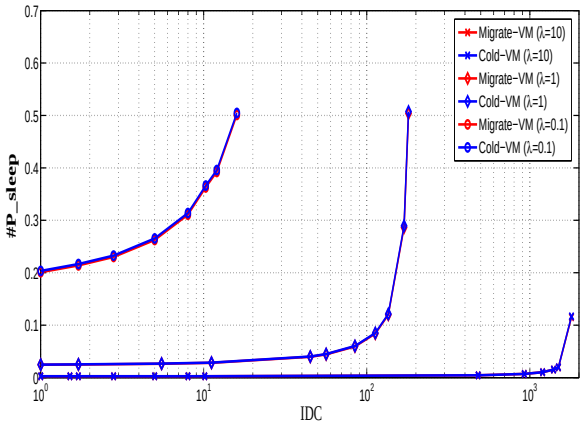


FIGURE 4.26 – *#P_sleep* en fonction de IDC (λ = 0.1, 1, 10)

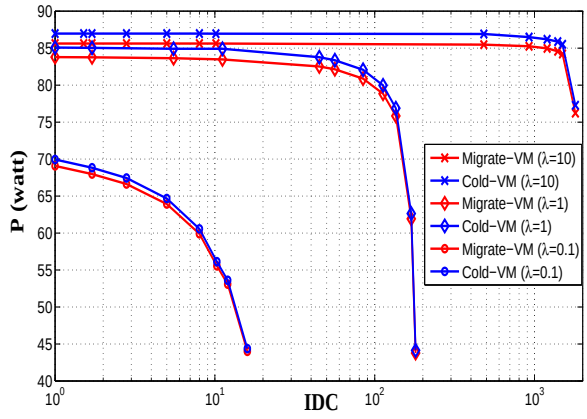


FIGURE 4.27 – Puissance moyenne consommée (P) en fonction de IDC (λ = 0.1, 1, 10)

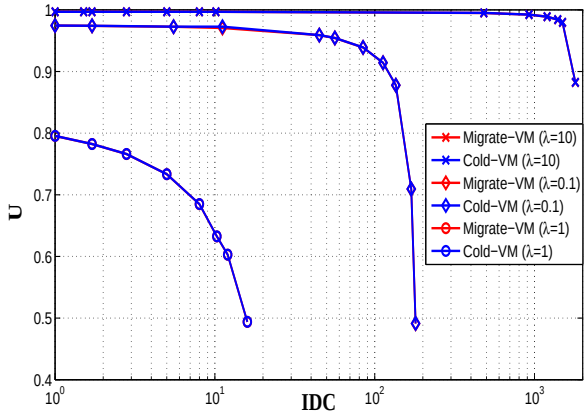


FIGURE 4.28 – Taux d'utilisation de la puissance (U) en fonction de IDC (λ = 0.1, 1, 10)

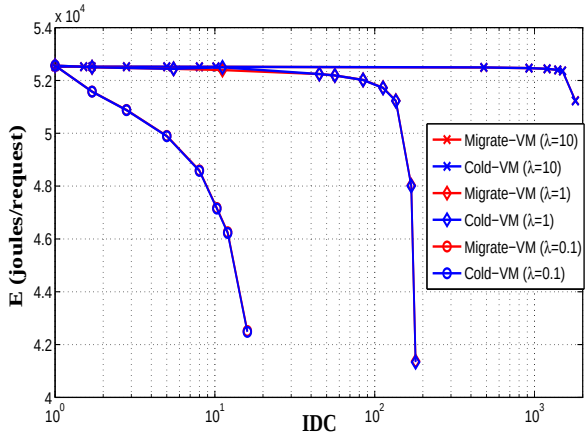


FIGURE 4.29 – Efficacité (E) en fonction de IDC (λ = 0.1, 1, 10)

4.9 Conclusion

Nous avons présenté dans ce chapitre un axe de recherche se rapportant à la sûreté de fonctionnement des systèmes virtualisés et en particulier la virtualisation des serveurs. Une modélisation d'un système de serveur virtualisé (SVS) basée sur des SRN non-markoviens a été proposée. Ce SVS composé d'un VMM et d'un VM et supportant des trafics sporadiques correspondant à des applications de données intensives. Le modèle SVS considère que le VMM et le VM sont soumis aux aléas du vieillissement logiciel et utilisent le rajeunissement logiciel comme méthode proactive pour lutter contre ce Vieillissement. Il intègre aussi un mécanisme de gestion de l'énergie basé sur temps adaptant les états d'énergie du PMC aux variations du trafic. L'analyse concerne deux types de rajeunissement VMM à savoir Cold-VM et Migrate-VM.
L'exploitation du modèle a été faite au travers la définition de plusieurs métriques mettant en évidence les aspects de performance ; de performabilité et d'efficacité énergétique telles que la disponibilité en régime permanent, la puissance moyenne consommée, le taux d'utilisation de l'énergie. L'analyse numérique permet de capturer l'impact de la variation de la charge de travail et le trafic en rafale sur les métriques de performance SVS. Les résultats obtenus confirment que pour les SVS soumis au vieillissement en fonction de la charge de travail et que le trafic devient plus en rafale, le SVS devient plus disponible et plus économe en énergie.

Chapitre 5

Modélisation des performances du commutateur virtuel avec arrivées groupées

Les réseaux des centres de données sont généralement composés de multiples périphériques réseau hétérogènes comme les commutateurs, les routeurs, les pare-feux, etc. La configuration des équipements utilisés dans les infrastructures réseau courantes dépend des fabricants. Une fois les équipements sont déployés, la mise à jour et l'évolution des réseaux s'avèrent difficiles. De plus, la variabilité du trafic et de la dynamique du réseau a poussé vers la nécessité d'avoir des réseaux programmables. C' est dans ce contexte que sont apparues deux approches (SDN et NFV) qui permettent de s'adapter à la nature dynamique des réseaux et augmenter l'agilité.
Dans ce chapitre, nous allons intéresser par l'un des composants du plan de données qui est le commutateur virtuel. Nous modélisons son architecture par un modèle d'attente afin d'évaluer ses performances.

Sommaire

5.1 Commutateur virtuel

5.1.1 Context

Dans l'architecture SDN / NFV, le commutateur virtuel (VS :virtuel switch) représente un nœud critique qui détermine les performances du réseau. Il se comporte comme un commutateur matériel. Il s'agit d'un logiciel intégré à l'hyperviseur qui permet aux machines virtuelles (VM) de s'intégrer au réseau. Il établit la communication entre des VMs existant sur le même hôte physique ou sur un hôte physique différent (Figure 5.1). Le commutateur virtuel est chargé de la réception, du traitement, de la commutation et de la transmission des paquets circulant dans le réseau. Il contient une ou plusieurs tables de flux conçues pour le transfert de trafic. Chaque table contient plusieurs entrées (flow entries) qui représente chacune une règle qui contient des informations sur les paquets (l'adresse de destination, le numéro du port, ...).
Lorsqu'un paquet arrive au commutateur, son en-tête est lu pour extraire l'adresse de destination. Cette adresse sera comparée à la table de flux pour déterminer le port de sortie adéquat. Le commutateur peut exercer certaines étapes supplémentaires, telles l'inspection profonde de paquets, cryptage, contrôle de la qualité de service, etc. Une fois que le port de sortie est trouvé, le paquet est transféré vers le port de sortie. Si l'adresse de destination n'existe pas dans la table de flux, le paquet sera transmis au contrôleur SDN, qui détermine le meilleur chemin et installe ainsi des règles du flux dans le commutateur.
Le contrôleur peut ajouter, modifier ou supprimer des règles dans les tables de flux, d'une façon réactive (comme réponse à l'arrivée d'un paquet) ou proactive (installer les règles à l'avance dans le commutateur). Les règles de transmission associées sont programmées via une API normalisée telle que OpenFlow (OF) [119]. OpenFlow est un protocole de communication entre le contrôleur et le commutateur dans un réseau SDN. De nombreuses implémentations utilisent le protocole OF. Open vSwitch (OvS) est la solution la plus populaire qui supporte le protocole OF. OvS est un commutateur compatible OF et standardisé par l'organisme Open Networking Foundation (ONF). Il fonctionne avec la plupart des systèmes d'hyperviseur et de conteneur, y compris Xen, KVM et Doker.
Un commutateur virtuel offre une grande flexibilité dans la gestion des ressources. En effet, plusieurs ressources comme les ports et les cœurs de processeur (CPU) peuvent être allouées ou désallouées de manière dynamique, permettant ainsi une utilisation optimale des ressources et une réduction de la consommation d'énergie. Par exemple, si un commutateur virtuel fait face à une charge de travail excessive, l'hyperviseur peut allouer des CPU supplémentaires pour atténuer les effets du conflit. Aussi dans le cas inverse où un commutateur virtuel utilise peu les ressources qui lui sont allouées, l'hyperviseur peut décider de dés-allouer une fraction de ses CPU. Ceci permet une réduction de la consommation électrique.

5.1.2 L'architecture d'un commutateur virtuel

Comme le commutateur physique, le commutateur virtuel comprend plusieurs cartes d'interface réseau (**NIC :** Network Interface Cards) qui constituent un total de N ports d'Entrée/Sortie (**E/S**). Il contient également un ensemble de **C CPU** chargés de traiter les paquets provenant des différents ports d'E/S. La figure 5.2 représente cette architecture interne. L'utilisation des NIC modernes permet à chaque port d'équilibrer la charge en répartissant le trafic entrant sur tous les CPU. Les paquets entrants provenant d'un seul port sont envoyés aux C files d'attente logiques, une par CPU, également appelées files d'attente RX (Réception). Chaque CPU est ainsi associé à N files d'attente RX indépendantes, une par port, traitées par interrogation (polling). Lorsque le polling d'une file d'attente RX commence, le CPU traite le premier paquet en ligne, puis il passe à la file d'attente RX suivante. Notez qu'il est possible d'activer le mode de traitement par lots en traitant les **M** premiers paquets de la file d'attente RX avant de passer à la file suivante. Le CPU doit connaître à l'avance le nombre (M) de paquets à interroger avant de commencer le traite-

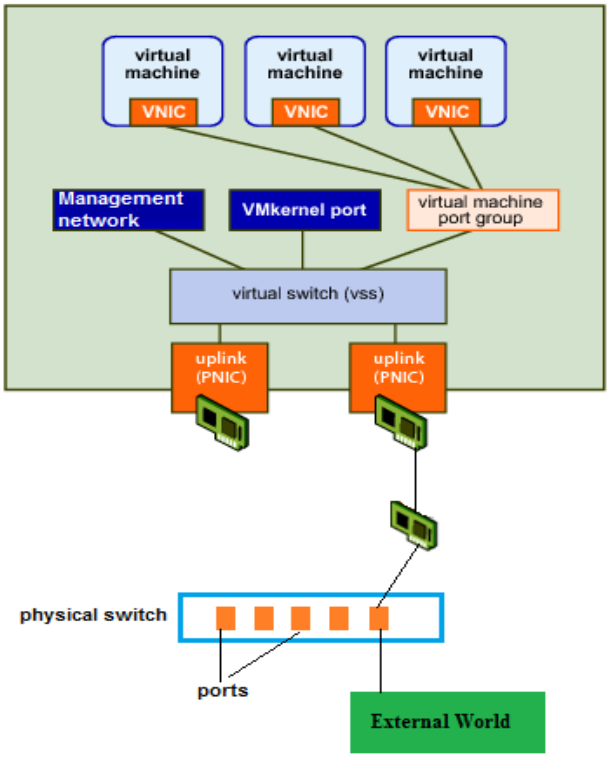


FIGURE 5.1 – Commutateur virtuel dans un réseau

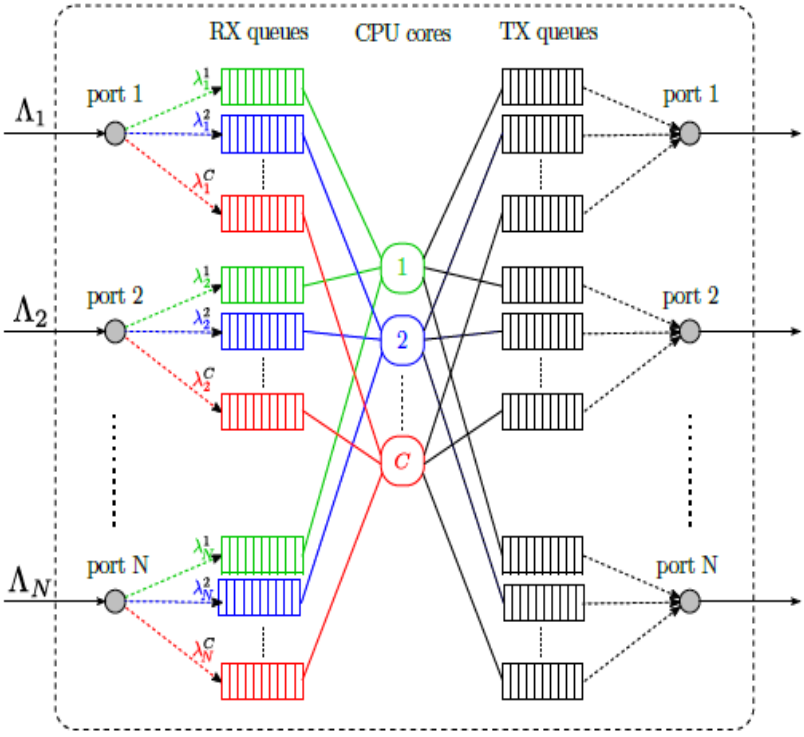


FIGURE 5.2 – L'architecture intérieure d'un commutateur virtuel

ment. Une fois que le CPU termine le traitement d'un paquet, ce dernier sera transféré de sa file d'attente RX vers une file d'attente logique TX (Transmission) associée au port de sortie.
Afin de répondre à une demande de service d'une manière rapide et efficace, différentes techniques, telles que Netmap, OpenOnload , PachetShader et DPDK ont été développées [120]. Les commutateurs OvS utilisent la bibliothèque DPDK (Data Plane Development Kit), qui s'exécute sur différentes architectures de CPU [121] et qui offre des pilotes en mode interrogation (polling) permettant l'accélération de la vitesse de traitement des paquets.
Le goulot d'étranglement d'un commutateur virtuel risque de se produire lors du traitement des paquets dans les files d'attente RX en raison des ressources limitées de la CPU (tailles de file d'attente, vitesses de fonctionnement des NIC et des CPU). En conséquence, les CPU ne peuvent pas traiter les paquets aussi rapidement que nécessaire, ce qui peut entraîner des rejets de paquets au niveau des ports d'entrée. La prévision d'un tel comportement peut être d'un grand intérêt pour étalonner les paramètres du commutateur, par exemple le nombre total C de cœurs de CPU attribués au commutateur. Nous concentrons donc nos efforts sur l'évaluation des performances du commutateur au niveau de la file d'attente RX.

5.2 Décomposition du système

L'architecture générale des commutateurs peut être divisée en C sous-systèmes indépendants, chacun d'eux contenant un CPU qui interroge N files d'attente RX indépendantes (N représente le nombre de ports). Chaque sous-système est identifié par une couleur distincte sur la figure 5.2. Par conséquent, ces sous-systèmes peuvent être étudiés indépendamment les uns des autres (Fig. 5.3). Nous nous intéressons maintenant à un tel sous-système simplement en tant que système de polling. Un tel système est caractérisé par un CPU j, j = 1 ; ... ; C, avec ses N files d'attente. Chaque file d'attente RX i est caractérisée par sa capacité finie **K**, un taux d'arrivée des paquets λ_i^j, ainsi par un temps de service moyen des paquets.
Pour des raisons de clarté, nous supprimons l'indice j dans toutes les notations et les équations.

5.2.1 Modèles de polling

Les modèles de polling sont utilisés pour représenter les systèmes de plusieurs files d'attente servies par un seul serveur dans un ordre défini [122] (Fig. 5.4. Un grand nombre des études existe dans la littérature sur ces systèmes qui n'a cessé de croître depuis la fin des années 1950. On peut citer [123], [124] et [125]. Les systèmes de polling peuvent être observés dans plusieurs domaines, comme les réseaux de commutation, les systèmes de trafic et de transport, les systèmes de fabrication et de maintenance [123]. Il existe de nombreuses variétés de modèles de polling qui peuvent être classés en fonction des disciplines de service, de l'existence ou non du temps de la commutation entre les stations (switch-over), de la capacité des buffers, de l'ordre dans lequel le serveur

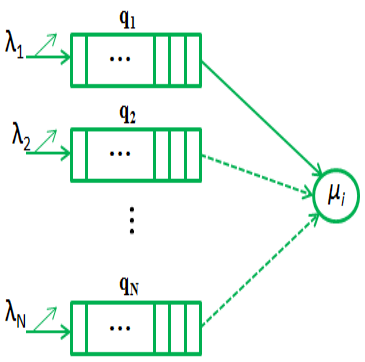


FIGURE 5.3 – Décomposition de l'architecture du commutateur

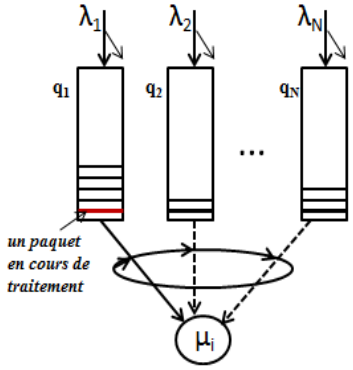


FIGURE 5.4 – modèle de polling

interroge les files d'attente, etc. Dans ce travail, on va s'intéresser par la discipline de service. En effet, la discipline de service détermine le nombre de clients servis par serveur à chaque tour. Les disciplines de service les plus utilisée sont exhaustive, gated et M-limited. Ils se diffèrent par les instants auxquels le serveur bascule d'une file d'attente à une autre [123].

— La discipline exhaustive, le serveur sert la file d'attente jusqu'à ce qu'elle soit vidée. Les clients arrivant à la file d'attente pendant la période de service sont également servis.

— La discipline gated, le serveur ne traite que les clients présents dans la file d'attente lors de son interrogation (instant d'interrogation). Ceux qui arrivent en file d'attente pendant le service sont mis de côté pour être servis lors de la prochaine période de service.

— La discipline M-limited, au plus M clients sont traités à chaque visite de la file d'attente.

La figure 5.4 présente un modèle de polling où on trouve N files d'attentes servies par un seul serveur à tour de rôle (round robin).

5.2.1.1 travaux annexes

L'étude des systèmes de polling est très complexe. Par conséquent, la résolution analytique exacte n'existe pas dans la littérature. Pour simplifier la résolution, éviter la croissance de l'espace d'états et le traitement des chaînes de Markov multidimensionnelles, plusieurs approximations ont été proposées. Les auteurs ont proposé une approche de décomposition du modèle en sous-modèles (le nombre de sous modèles est égale au nombre de files d'attente dans le modèle). Les auteurs ont fait recours au serveur avec vacances (server with vacation). Ainsi, la file considére le serveur comme étant parti en vacance. Le temps de traitement des autres files (le temps d'absence de serveur d'une file d'attente donnée) est un temps de vacances (vacation times) pour une file donnée.
Chaque sous-modèle a été modélisé par une chaîne de Markov induite. Cette méthode consiste à choisir une séquence d'instants 1,2,3,....n (déterministe ou aléatoire) pour décrire l'état du système. Cela ramène à une chaîne markovienne et homogène ce qui facilite l'analyse.
Dans [126] et dans [127], les auteurs ont représenté les files d'attente par des files de type M/G/1/N. Chaque file a été étudiée aux instants de polling. Dans [126], la discipline de service exhaustive a été étudiée, alors que dans [127] celle M-Limited qui a été analysée. L'analyse est basée sur la période d'occupation (busy period), c'est la période pendant laquelle le serveur n'est pas en vacances, et aussi sur l'hypothèse que les périodes de vacances sont indépendantes. La transformation de Laplace-Stieltjes a été utilisée pour déterminer la probabilité stationnaire. L'expression de la longueur de la file d'attente, la probabilité de blocage ainsi que le temps d'attente ont été donnée.
Dans [128], les auteurs ont proposé une méthode analytique pour calculer les performances d'un système de polling qui est caractérisé par une discipline de service gated 1-limited et un temps de

commutation non nul. La détermination des probabilités d'état de la chaîne de Markov est faite par un calcul des transformations de Laplace-Stieltjes et des techniques d'approximation à deux moments. Une technique itérative appelé technique de point fixe a été utilisée pour trouver la solution. Les résultats numériques tels que le temps moyen d'attente et la probabilité de blocage ont montré que le modèle n'est pas précis pour des temps de commutation faibles et des grandes capacités de la file d'attente (plus de 10 clients). La technique de point fixe développée dans le travail de Tran-Gia et Raith [128] a été étendue dans [129] pour étudier les systèmes de polling avec exhaustive M- limited.
Une étude plus générale a été présentée dans [130], où les auteurs ont réduit le nombre d'hypothèses tel que l'évaluation du modèle pendant les périodes de service (busy). Ils ont étudié les trois types de la discipline de service (exhaustive, gated, limited). L'approche suivie repose sur la résolution d'un système de plusieurs équations numériques. Des formules numériques ont été données pour exprimer le débit, la probabilité de perte et le temps d'attente pour une file d'attente particulière choisie arbitrairement. Cependant, vu la complexité des expressions, l'utilisation d'un logiciel de calcul performant a été nécessaire.
La plupart des travaux ont analysé chaque file d'attente à des instants particuliers. Aussi ils ont fait recours à la transformation de Laplace-Stieltjes pour évaluer les modèles. Toutefois ces hypothèses ne représentent pas des situations pratiques.
Par exemple, les transformations de Laplace-Stieltjes peuvent ne pas s'adapter à un nombre important de files d'attente ou à leur capacité. Pour répondre à ses contraintes, d'autres solutions ont été proposées. Dans [131], la modélisation basée sur des chaînes de Markov à temps continu a été proposée pour représenter les différents états d'une file d'attente. L'auteur a bien détaillé les états des vacances du serveur. Cependant, la résolution de chaque chaîne de Markov n'est pas facile et nécessite l'utilisation de méthodes numériques telles que la sur-relaxation successive (SOR) ou la méthode de Gauss-Seidel.
Le travail du Gallardo et al. [132] ont représenté chaque file d'attente par une chaîne de Markov à temps continu de type M/M/1/K. Chaque file d'attente a une capacité finie servie à tour de rôle. Les paquets arrivent au système selon un processus de Poisson λ. Le temps de service suit une loi exponentielle de taux μ, indépendamment d'un client à l'autre. Les autres paramètres de la chaîne comme le taux de vacance, la probabilité de rester sur la même file ou visiter une autre file ont été calculés en utilisant la technique de point fixe. Après la détermination de la probabilité stationnaire, différents paramètres de performance ont été évalués tels que la longueur de la file d'attente, la probabilité de blocage et le temps de séjour.

5.3 Files d'attente avec vacance

Les systèmes de files d'attente avec vacances représentent une classe très importante de files d'attente où le serveur n'est pas disponible pendant certains intervalles de temps. Pendant ces périodes qui sont appelées périodes de vacances, les clients peuvent entrer dans le système, mais attendre dans une file d'attente (s'il existe des places) pour être servis une fois les vacances terminées et le serveur reprend le service. Il est noté que durant ces vacances, le serveur pourrait être utilisé pour une autre tâche afin d'améliorer son utilisation. La durée de vacances peut être dépendante ou indépendante du durée de service. Dans plusieurs modèles, les périodes de pannes ou de réparation peuvent être vues comme des périodes de vacances.
On distingue deux types de vacances du serveur. Le premier type est connu sous le nom d'un service exhaustif où le serveur prend des vacances lorsque le système est vide. Après la période de vacance, deux politiques existent.Le serveur peut reprendre les vacances s'il ne trouve pas des clients dans le système (le serveur ne reprend son activité que s'il existe au moins un client dans le système). On dit dans cette situation qu'on a un système avec *vacances multiples*. La deuxième politique, le serveur prend une seule vacance aprés chaque période d'activité (Si aucun client n'est arrivé, le serveur attend la première arrivée). Cette politique est connue sous le nom de *vacance*

unique. Le deuxième type de vacance, le serveur prend une vacance même s'il existe des clients dans le système, on dit que le service est non-exhaustif.
Les files d'attente avec vacances ont été étudiées pendant plus de deux décennies pour modéliser et analyser les systèmes informatiques, les réseaux de communication, les systèmes de fabrication et de production et bien d'autres. La littérature est vaste et riche. Levy et Yechiali [133] ont été les premiers à étudier ces systèmes. Ils ont proposé une file M/G/1 en supposant que le service est exhaustif. Deux modèles ont été étudiés un modèle avec vacance unique et un autre avec vacances multiples. Des transformations de Laplace-Stieltjes de la période d'occupation, de la période de vacances et du temps d'attente ont été dérivées, ainsi des fonctions génératrices du nombre d'unités dans le système ont été calculées pour les deux modèles. Les résultats ont été comparées après afin de trouver des temps de vacances optimaux.
Des variantes et des généralisations ont été présentées après dans les travaux de de Doshi [134], Takagi [135] et Tian et Zhang [136]. Dans ces études, les auteurs ont supposé que le serveur est inactif pendant les vacances.
Servi et Finn [137] ont analysé une file d'attente M/M/1 avec vacances où le serveur travaille à un rythme différent durant les vacances au lieu d'arrêter complètement le service. Ainsi, la période de vacances devient une période de fonctionnement à vitesse réduite. Les paramètres de performances comme la distribution du nombre de clients dans le système et le temps de séjour en régime permanent ont été obtenus. Ces résultats peuvent être appliqués afin d'analyser un système à plusieurs files d'attente dont le taux de service est l'une des deux vitesses de service. Le serveur se déplace vers une autre file d'attente pendant les vacances. Les auteurs ont appliqué les résultats dans le réseau d'accès optique à multiplexage par répartition en longueur d'onde (WDM).

5.3.1 Les modèles de files d'attentes avec arrivées groupées

Lors de l'étude des problèmes classiques de la théorie des files d'attente, on supposait que les clients arrivaient un par un. Cependant, dans plusieurs situations rencontrées dans la pratique (tels que les systèmes informatiques, les réseaux de communication, les systèmes de production, les systèmes de transport (trafic maritime et aérien, ...), les systèmes de fabrication (dans l'industrie électriques et électronique),...), les clients arrivent en groupes de taille aléatoires. Ces situations d'attente peuvent être représentées par des modèles appelés modèles avec arrivées par groupes (queueing systems with batch (bulk) arrivals). Ces derniers ont fait l'objet de recherches approfondies au cours des dernières décennies, on peut citer les travaux de Medhi [138], Neuts[139], ...
Les auteurs dans [140] ont modélisé le fonctionnement des serveurs dans un centre de données. Ces serveurs sont caractérisés par un temps de démarrage et une vitesse de service variable afin de réduire la consommation de l'énergie. La vitesse de service est proportionnelle au nombre des requêtes dans le système. Ainsi, un modèle de file d'attente $M^X/M/1/SET-VARI$ a été présenté.. Les auteurs ont dérivé une expression de la fonction de génération de probabilités du nombre d'emplois dans le système. La transformée de Laplace-Stieltjes (LST) de la distribution du temps de séjour a été obtenue sous forme de séries impliquant des matrices de dimension infinie. Des exemples numériques ont été présenté. afin de montrer le compromis entre le temps moyen de séjour et la consommation d'énergie.
Yao et al. ont proposé un système $M^X/M/1$ pour traiter le problème du capacité du contrôleur dans une architecture SDN OpenFlow. Cette capacité peut être définie comme le nombre de commutateurs qu'un contrôleur peut gérer. Les demandes entrantes ont été modélisées par des arrivées groupées avec une distribution de Poisson. Les résultats obtenus ont été utilisés pour évaluer la capacité du contrôleur. Aussi, les auteurs ont étendu l'étude d'un seul contrôleur à plusieurs contrôleurs. Tous ces résultats sont significatifs pour le déploiement du réseau OpenFlow à grande échelle.
Pour les systèmes de files d'attente avec arrivées groupées à capacité finie, Monfield et Tan [141] ont étudié le système $M^X/M/m-s$ afin d'étudier les performances d'un controlleur de communi-

cation. Ce dernier stocke les paquets arrivants de l'hôte avant les envoyées vers le commutateur. L'arrivée des paquets est modélisée par des arrivées groupées. Les probabilités d'états sont déterminées par une récursivité numérique et utilisées pour déterminer les probabilités de blocage et la distribution des temps d'attente en fonction de deux stratégies d'acceptation des groupes. Les résultats numériques ont montré l'impact de la taille moyenne des groupes ainsi que la distribution du groupe sur les performances du système. Aussi, la stratégie d'acceptation a une influence significative. Le choix de stratégie dépends de l'application et la nature des paquets.
Dragovic et al [142] ont évalué les performances du port fluvial en utilisant des modèles de file d'attente $M^X/M/n/m$ avec des arrivées par lots, une distribution de service exponentielle et des zones d'attente finies. Les auteurs ont considéré deux stratégies d'acceptation de lots (acceptation totale ou partielle des groupes). Des résultats des probabilités de blocage des lots et celles de blocage d'une arrivée arbitraire dans les cas non stationnaires et stationnaires ont été obtenus.

5.3.2 Modèles de files d'attente avec arrivées par groupes et avec vacance

Des efforts considérables ont été consacrés à l'étude des modèles d'attente avec arrivées par groupes et avec vacance. Baba[143] a étudié pour la première fois les systèmes de files d'attente $M^X/G/1$ à vacances multiples. À la fin de chaque période de vacances, si le serveur ne trouve pas un client dans la file, il repart en vacances. Il a donné la fonction génératrice du nombre de clients dans le système juste après le départ d'un client et la transformée de Labplace du temps d'attente d'un client, en utilisant la technique des variables supplémentaires.
Choudhury [144], a analysé le système $M^X/G/1$ à vacances multiples, il a étudié le comportement de la distribution de la longueur de la file aux instants arbitraires et aux instants de départ des clients par une approche analytique, il a obtenu aussi la fonction génératrice de la longueur de la file aux instants de départ et quelques mesures de performances. Xu et al. ont étudié le système $M^X/M/1$ avec vacances unique. Ils ont utilisé la méthode analytique matricielle pour dériver la fonction génératrice des probabilités (PGF)en régime stationnaire. À partir de cette fonction, ils ont obtenu la décomposition stochastique de la longueur du système stationnaire qui indique la relation avec celle de la file d'attente $M^X/M/1$ classique sans vacances.
Baba [145] a analysé le système $M^X/M/1$ avec vacances multiple où le serveur sert les clients à deux taux de service différents. À partir de la chaîne de Markov bidimensionnelle, il obtient la matrice de transition qui a une forme triangulaire quasi supérieure. Ainsi, la fonction de génération de probabilités de la distribution de longueur du système a été dérivée, en utilisant une méthode analytique matricielle. Il a obtenu la structure de décomposition stochastique de la longueur du système et à partir du quelle la relation entre $M^X/M/1$ avec vacance multiple et $M^X/M/1$ sans vacance a été trouvée.
Les auteurs dans [146] ont étudié une file d'attente avec arrivées groupées, service par groupe, capacité finie et avec vacances uniques et multiples. La distribution du nombre de clients dans la file d'attente en régime permanent à différents instants (à la fin du service, à la fin des vacances, au départ, et avant l'arrivée) a été obtenue. D'autres mesures de performance telles que le temps d'attente moyen, la probabilité que le serveur soit occupé, les probabilités de blocage, sont discutées avec quelques résultats numériques. Ils ont étudié aussi l'effet de certains paramètres du modèle sur les principales mesures de performance.

5.4 Contributions

Notre travail est basé sur le modèle présenté dans [132]. Les auteurs ont présenté un modèle analytique basé sur les files d'attente afin d'évaluer les performances d'un commutateur virtuel

avec plusieurs cartes d'interface réseau (NIC ou des ports) et plusieurs cœurs de processeur (CPU) (Fig.5.2). Chaque CPU sert un ensemble des ports, d'où la décomposition du système en sous systèmes. Chaque sous système est représenté par un système de polling (interrogation) où un serveur (CPU) qui sert un ensemble de files d'attente (Chaque file d'attente représente un port) dans un ordre défini. Cependant, afin d'éviter la croissance combinatoire de l'espace d'état associée à ces modèles, le système associé à chaque CPU est décomposé en plusieurs files d'attente avec vacances pour capturer les interactions entre les files d'attente.
Les auteurs ,dans ce travail, ont supposé que les paquets arrivaient séparément selon un processus d'arrivées de Poisson. Cependant, une telle hypothèse n'est pas réaliste car plusieurs travaux ont montré que les paquets arrivent par groupes et non pas séparément. C'est pour ces raisons que nous traitons dans cette thèse un processus d'arrivée par groupes. Dans les sections suivantes, nous présenterons le modèle de markov décrivant le traitement des paquets par un CPU du commutateur virtuel, les différents paramètres. Après, nous donnerons les différentes étapes d'analyse du système ainsi que les mesures de performances.

5.5 Le modèle de Markov

On part de l'idée de décomposition du système de polling en un ensemble des files d'attente indépendantes avec des arrivées groupées et vacances du serveur. Durant la période de vacances, le serveur est occupé par le service des paquets dans les autres files. La figure 5.3 montre cette décomposition. La figure 5.5 illustre un exemple particulier. Le serveur (CPU) traite actuellement un paquet dans la file d'attente RX i et que toutes les autres files d'attente sont en attente. Lorsque le serveur termine le traitement d'un paquet, il passe à la file suivante i+1. Ainsi, le paquet suivant de la file i est mis en attente. La période entre la fin de service d'un paquet et le retour à la file i pour servir un autre est considérée comme un temps de vacance. Par conséquent, la file d'attente i commence ses vacances, et en même temps, la file i+1 termine ses vacances et déclenche le traitement de son premier paquet en attente.
La description détaillée du modèle est donnée ci-dessous :

— Chaque file a une capacité finie K;

— Des groupes des paquets arrivent selon un processus de Poisson de taux $\lambda > 0$. Le nombre de clients par groupe est une variable aléatoire X strictement positive.
On note g_m= P(X=m) =$Pr\{$ le groupe d'arrivées contient m paquets, m ≥ 1 $\}$.
On note $\lambda_m = \frac{\lambda}{g_m}$, le taux d'arrivées des groupes comportant m paquets. Ainsi, λ est le taux d'arrivée composite de tous les lots est égale à $\sum \lambda_m$.

— Les paquets sont servis individuellement, les durées des services étant indépendantes et distribuées suivant une loi exponentielle de taux μ.
Les arrivées des clients ne forment pas un processus de Naissance et de Mort. Cependant, le

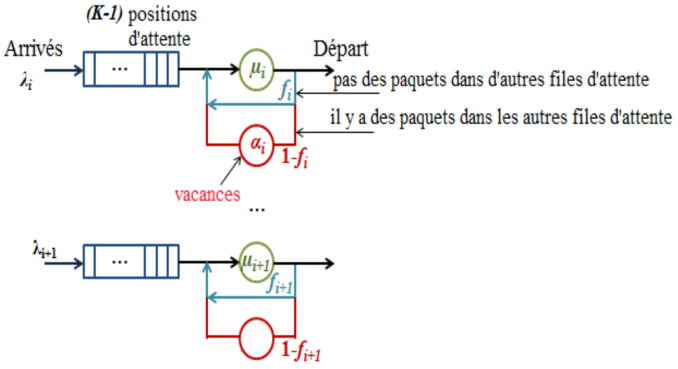


FIGURE 5.5 – Décomposition d'un sous-système en N files d'attente

système est Markovien, puisque le comportement futur dépend uniquement de la situation présente.

— À chaque tour, le serveur sert un paquet pour chaque file d'attente, puis passe à la file d'attente suivante. Les temps de service des autres files sont vus comme un temps de vacances. La période de vacances commence à la fin de chaque temps de service d'un paquet dans une file particulière et dure jusqu'à ce que le serveur revienne à cette file. Le temps de vacance est distribué de manière exponentielle avec un taux α_i.
Nous supposons que le flux des arrivées, le temps de service et le temps de vacances sont mutuellement indépendants.

— Si les autres files d'attente sont vides et il existe des paquets dans la file servie i, le serveur traite le paquet suivant et ne prend pas des vacances. La probabilité de cet événement est égale à f_i.

— S'il existe de paquets dans les autres files, le serveur passe à la file suivante avec une probabilité 1- f_i.

— La discipline de service adoptée dans ce travail est FIFO (Fist In First Out).

L'état du système à l'instant t peut être décrit par le processus M(t)=(L(t), J(t))$_{(t\geq 0)}$, où
L(t) : le nombre de clients dans la file donnée i à l'instant t,

$$J(t) = \begin{cases} \text{P} & \text{si le serveur traite un paquet dans la file i (Processing).} \\ \text{R} & \text{si le premier paquet dans le buffer est en attente (le serveur est en vacance)(Ready).} \\ \text{E} & \text{si la file i est vide (Empty).} \\ \text{F} & \text{si toutes les files sont vides (on a un système vide)(Full Empty).} \end{cases}$$

Il est clair que le processus (L (t), J (t)) presente une chaîne de Markov à deux dimensions avec un espace d'état

$$\Omega = \{(0,\text{F}),(0,\text{E})\} \cup \left\{(k,j) | k = 1\ldots,\text{K}; j = \text{P},\text{R}\right\}.$$

5.5.1 Chaine de Markov associéé à chaque file d'attente

La chaîne de Markov associé à chaque file d'attente i du système de polling est représentée par la figure 5.6. L'état (k, P), k = 1..K, correspond à la file d'attente i avec k paquets et le CPU traite un paquet (P), donc le CPU est affecté à la file d'attente RX i. L'état (k, R), k = 1..K, correspond également à la même file d'attente i avec k paquets, mais avec un premier paquet qui est mis en attente (le paquet est prêt à être traité (R)), ainsi le CPU est affecté à une autre file RX. L'état (0, E) correspond à une file d'attente i vide (E) mais pas à un système complètement vide, car il existe des paquets dans les autres files (le CPU est affecté à une autre file d'attente RX non vide). Finalement, l'état (0, F) correspond à un système (F) vide, ce qui signifie que toutes les files d'attente sont vides et le CPU est inactif. Par conséquent, les états ((0, E) et (k, R)) correspondent à une file d'attente avec un serveur en vacances, les états (k, P) représentent une file d'attente avec un service de traitement, et l'état (0, F) correspond à un système vide. De n'importe quel état de cette chaîne (sauf (K, P) et (K, R)), nous pouvons passer à l'état adjacent avec taux d'arrivées des groupes comportant m paquets λ_m, à condition que k+m soit inférieur ou égale à K. Si k+m ≥ K, il existe deux politiques d'acceptation ; soit le groupe sera totalement rejeté, soit on remplit les positions disponibles et les paquets restants seront rejetés.
Après un temps de traitement $\frac{1}{\mu}$, il y a des possibilités : Si toutes les autres files d'attente $j \neq i$ sont vides, le CPU est immédiatement réaffecté à la file d'attente i avec une probabilité f_i, ainsi, le CPU ne part pas en vacances. Dans ce cas, le processus markovien passe de l'état (k,P) vers l'état (k-1, P) (ou (0, F) si k=1). Dans le cas contraire, où au moins une des autres files d'attente ($j \neq i$) n'est pas vide, le CPU part en vacance avec une probabilité 1- f_i. Par conséquent, le processus markovien passe de l'état (k, P) vers l'état (k-1, R) (ou (0, E) si k=1).
À partir de n'importe quel état (k, R), k = 1..K, on peut atteindre l'état (k, P) avec un taux α_i correspondant au temps de vacances (le CPU revient à la file d'attente i). À partir de l'état (0, E), on peut

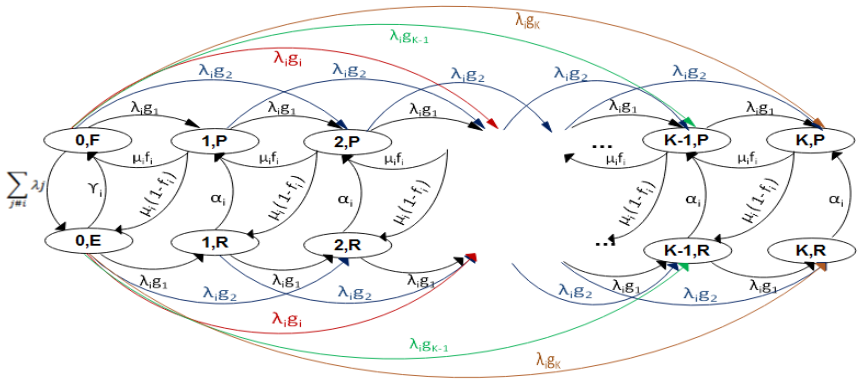


FIGURE 5.6 – Chaîne de Markov à temps continu associée à la file d'attente i.

atteindre l'état (0, F) avec un taux γ_i. En effet, cette transition correspond à un système devenant complètement vide, partant d'un état où la file d'attente i est vide. Enfin, à partir de l'état (0, F) correspondant à un système complètement vide, nous pouvons atteindre l'état (0, E) si le premier groupe d'arrivées entre dans une autre file d'attente $j \neq i$, c'est-à-dire avec un taux $\sum \lambda_j$.

5.5.2 Estimation des paramètres de la chaîne de Markov

Dans ce qui suit, nous donnons les expressions de probabilité manquantes f_i, α_i et γ_i, associés à une file d'attente donnée i.
Nous pouvons définir la probabilité f_i comme la probabilité que toutes les autres files d'attente ($j \neq i$) soient vides, sachant que le CPU est en cours de traitement d'un paquet de la file d'attente i (on est dans l'état (k, P)).

$$f_i = \prod_{j \neq i} \xi_j \tag{5.1}$$

Où ξ_j est la probabilité d'être dans l'état (0, E) de la chaîne de Markov j, sachant qu'elle ne peut être ni dans l'état (0, F) (car le système n'est pas totalement vide) ni dans l'état (k, P) (car le CPU traite un paquet de la file d'attente i).

$$\xi_j = \frac{\pi_j(0,\mathrm{E})}{\pi_j(0,\mathrm{E}) + \sum_{k=1}^{\mathrm{K}} \pi_j(k,\mathrm{R})}. \tag{5.2}$$

Pour le paramètre α_i, nous estimons le temps de vacances moyen $\frac{1}{\alpha_i}$. Comme le temps moyen de vacances de la file i correspond au temps moyen de traitement d'un paquet de chaque file non vide $j \neq i$, nous pouvons écrire :

$$\frac{1}{\alpha_i} = \sum_{j \neq i} \frac{1}{\mu_i} \times \frac{1-\xi_j}{1-f_i} = \frac{1}{1-f_i} \sum_{j \neq i} \frac{1}{\mu_i}(1-\xi_j) \tag{5.3}$$

Afin d'estimer le taux de transition γ_i entre l'état (0, E) et l'état (0, F) de la chaîne de Markov i, nous résolvons un ensemble des équations en faisant une coupe à l'état (0, F). Nous obtenons :

$$\gamma_i = \frac{\lambda_i \pi_i(0,\mathrm{F}) - \mu_i f_i \pi_i(1,\mathrm{P})}{\pi_i(0,\mathrm{E})} \tag{5.4}$$

5.5.3 Technique de point fixe

D'après les équations (5.1, 5.3, 5.4), nous remarquons que les expressions des paramètres de la chaîne de Markov associée à une file d'attente donnée i (f_i, α_i et γ_i) dépendent de la solution stationnaire de la chaîne de Markov i et de la solution stationnaire des autres chaînes de Markov j.

En conséquence, nous utiliserons une technique itérative appelée technique à point fixe résumée par l'algorithme 1.
Premièrement, nous donnons des valeurs intiales de paramètres (π_i, f_i, α_i et γ_i) appelées des points initiaux et considérés comme une première solution. Puis nous procédons par itérations au cours desquelles, une succession de solutions approximatives, est déterminée. Ces valeurs se rapprochent graduellement de la solution cherchée. L'exécution de la boucle est répétée jusqu'à ce qu'une condition soit fausse. Dans notre cas, la boucle principale de l'algorithme est répétée jusqu'à ce qu'un critère de convergence donné soit atteint (la différence entre deux paramètres de deux itérations successives est très faible (par exemple, moins de 10^{-4}).

Algorithm 1: Technique à point fixe

Input : Paramètres système : K, μ_i, λ_i pour chaque file d'attente i
Output: Solution stationnaire π_i pour chaque file d'attente i

```
Initialisation π_i, f_i , α_i and γ_i pour chaque file d'attente i;
while condition est fausse do
    for Pour chaque file d'attente i ∈ [1..N] do
        Calculer la probabilité fi;
        Calculer le taux de transition α_i;
        Calculer le taux de transition γ_i;
        Résoudre la chaîne de Markov associée à la file d'attente i et Calculer la probabilité
         stationnaire π_i;
    end
end
```

5.5.4 Politique d'acceptation des groupes

Puisque la capacité de la file d'attente est finie, aucun paquet ne peut y entrer lorsque la file est pleine. Cela introduit un blocage dans la file d'attente et des pertes de paquets à l'entrée du système. Comme les arrivées sont groupées, on distingue deux types de stratégies d'acceptation des groupes.

— **Stratégie 1 :** Si la taille du groupe de paquets entrant dans le système est inférieure au nombre de places disponibles dans le buffer, le groupe sera accepté sinon il sera totalement rejeté. Cette stratégie est appelée *la stratégie d'acceptation totale des groupes.*

— **Stratégie 2 :** Si un groupe de m paquets arrivant dans le système trouve k paquets dans le buffer, il existe deux possibilités. Si $m \leq \mathrm{K} - k$, la totalité du groupe sera accepté. Si m > K-k, K-k paquets du groupe remplissent les positions disponibles et les paquets restants seront perdus. Cette stratégie est appelée *stratégie d'acceptation partielle des groupes*

5.6 Mesures de performance

Afin d'obtenir des informations sur les comportements des phénomènes modélisés, on s'intéresse aux mesures de performance (temps de réponse d'un service, taux d'utilisation d'une source, probabilité de perte,...). Lorsque les probabilités d'état sont connues, les mesures de performance peuvent être facilement obtenues. Dans ce travail, les performances étudiées en régime stationnaire sont :

— q_i : Le nombre moyen de paquets présents dans la file d'attente i.

$$q_i = \sum_{k=1}^{\mathrm{K}} k(\pi(k,\mathrm{P}) + \pi(k,\mathrm{R})) \tag{5.5}$$

— r_i : La durée moyenne du séjour (ou de réponse) est le temps moyen passé par un paquet dans la file d'attende i (l'intervalle de temps qui sépare son entrée et sa sortie du système). C'est un indice principal pour caractériser les performances du modèle. En se basant sur la formule de Little, le temps de séjour est égale au nombre moyen de paquets dans le système divisé par leur taux d'arrivée effectif, d'où l'expression donnée par l'équation suivante :

$$r_i = \frac{q_i}{\lambda_i^*} \tag{5.6}$$

Où λ_i^* est le taux d'arrivée effectif (le nombre des paquets entrant dans le système).

— b_i : La probabilité de blocage peut être définie aussi comme la probabilité que le système soit dans un état bloquant. Puisqu'on a un système de file d'attente à capacité finie, les nouveaux groupes (ou une partie) seront rejetés s'ils arrivent alors que la file est déjà pleine.

5.7 Taille maximale du groupe est fixe

Dans cette partie, nous supposons que le groupe des paquets arrivant avec une taille maximale fixe. Nous rappelons la probabilité qu'un groupe d'arrivées contient m paquets est g_m, m=1...B. B est la taille maximale du groupe d'arrivées et B < K. Ainsi on a $\sum_{m=1}^{B} g_m$=1.
Comme nous l'avons mentionné dans la description du modèle, les arrivées des paquets ne forment pas un processus de Naissance et de Mort. Par contre, ils représentent un processus quasi naissance et mort (Quasi Birth and Death : QBD).

5.7.1 cas particulier : la taille maximale du groupe d'arrivées est égale à 2

Dans cette partie, nous traitons un cas particulier où la taille maximale d'un groupe est égale à 2. Donc, nous avons deux possibilités soit un seul paquet qui vient d'arriver à la file d'attente i (le lot contient un seul paquet) avec la probabilité g_1, soit deux paquets sont arrivés en même temps (le groupe contient 2 paquets) avec une probabilité g_2. La figure 5.7 présente le diagramme de transition de la file d'attente i avec une taille maximale du groupe égale à 2. Il est bien clair qu'il y a des transitions entre les niveaux non adjacents. Ainsi par simple ré-ordre des états, en regroupant des états de tel sorte on limite les transitions entre les états qui sont dans le même niveau ou entre deux niveaux adjacents. Nous avons obtenu le diagramme représenté dans la figure 5.8.

5.7.2 Probabilités de transition stationnaires

En se basant sur la figure 5.8, la matrice de transition Q est donnée par

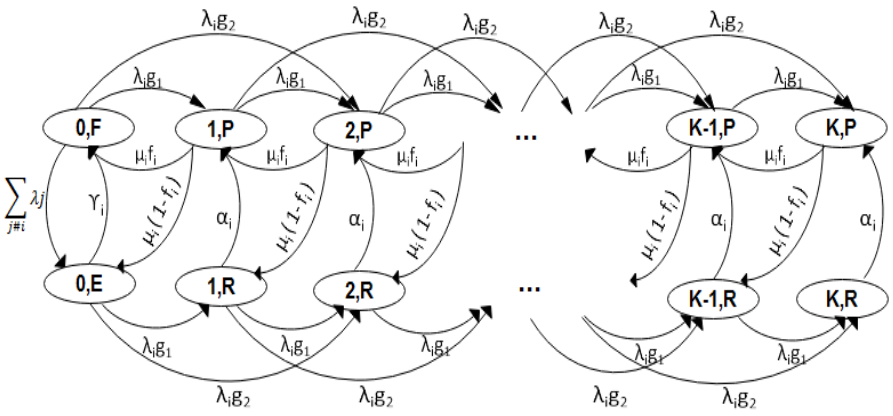


FIGURE 5.7 – Le diagramme de transition associé à la file d'attente i où la taille maximale du groupe d'arrivées est égale à 2 (g_1+g_2=1).

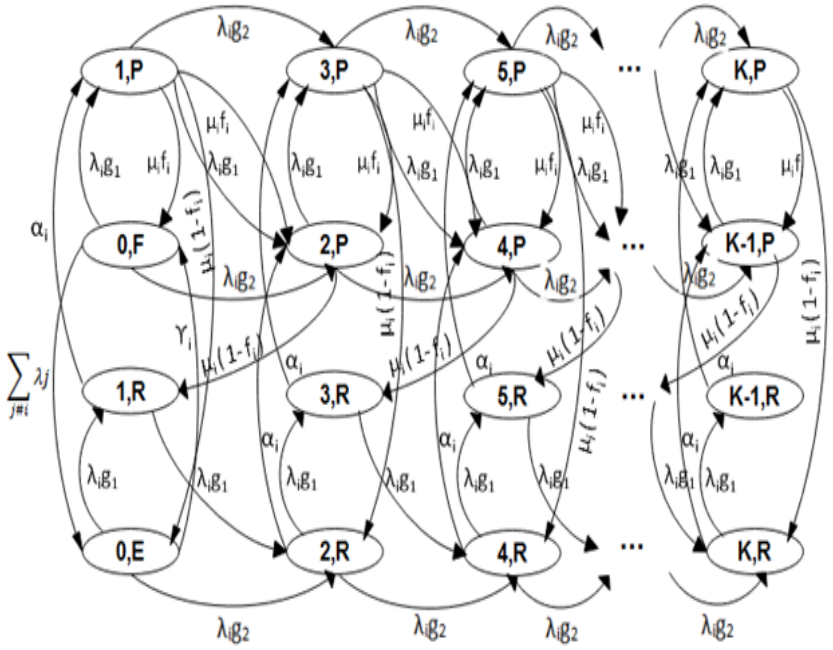


FIGURE 5.8 – Présentation alternative du diagramme de transition de la file d'attente i $M^X/M/1/K$ avec une taille maximale du groupe égale à 2.

$$Q = \begin{pmatrix} \hat{L} & \hat{F} & 0 & 0 & 0 & \dots \\ \hat{B} & L & F & 0 & 0 & \dots \\ 0 & B & L & F & 0 & \dots \\ 0 & 0 & B & L & F & \dots \\ 0 & 0 & 0 & B & L & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & L & F_{\Psi-1} \\ \dots & \dots & \dots & \dots & B_\Psi & L_\Psi \end{pmatrix}$$

Où tous les éléments sont des matrices 4*4.

$$\hat{L} = \begin{pmatrix} -(\mu_i+\lambda_i) & \mu_i f_i & 0 & \mu_i(1-f_i) \\ \lambda_1 & -(\lambda_i + \sum_{i\neq j}\lambda_j) & 0 & \sum_{i\neq j}\lambda_j \\ \alpha_{i,1} & 0 & -(\alpha_i+\lambda_i) & 0 \\ 0 & \gamma_i & g_1\lambda_i & -(\gamma_i+\lambda_i) \end{pmatrix}$$

$$\hat{F} = \begin{pmatrix} g_2\lambda_i & g_1\lambda_i & 0 & 0 \\ 0 & g_2\lambda_i & 0 & 0 \\ 0 & 0 & g_2\lambda_i & g_1\lambda_i \\ 0 & 0 & 0 & g_2\lambda_i \end{pmatrix}, \hat{B} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \mu_i f_i & 0 & \mu_i(1-f_i) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$L = \begin{pmatrix} -(\mu_i+\lambda_i) & \mu_i f_i & 0 & \mu_i(1-f_i) \\ \lambda_{i,1} & -(\mu_i+\lambda_i) & 0 & 0 \\ \alpha_i & 0 & -(\alpha_i+\lambda_i) & 0 \\ 0 & \alpha_i & \lambda_{i,1} & -(\alpha_i+\lambda_i) \end{pmatrix} L_\Psi = \begin{pmatrix} -\mu_i & \mu_i f_i & 0 & \mu_i(1-f_i) \\ g_1\lambda_i & -(\mu_i+g_1\lambda_i) & 0 & 0 \\ \alpha_i & 0 & -\alpha_i & 0 \\ 0 & \alpha_i & g_1\lambda_i & -(\alpha_i+g_1\lambda_i) \end{pmatrix}$$

$\hat{B}=B_\Psi=B$;

$\hat{B}=F_{\Psi-1}=F$

$g_1+g_2=1$

Ψ c'est le nombre de niveaux.

Les processus QBD ont des propriétés structurelles intéressantes qui peuvent simplifier le calcul des probabilités stationnaires. Comme mentionné précédemment dans le chapitre 3, il existe dans la littérature, une panoplie de travaux ayant l'objet la recherche et le calcul du vecteur de probabi-

lité d'un processus QBD.
Dans ce travail, nous avons utilisé la méthode proposée dans [92] pour résoudre le processus QBD à espace d'états fini. Cette méthode peut être appliquée à plusieurs systèmes de file d'attente. Les auteurs ont montré que le vecteur de probabilité stationnaire a une représentation géométrique matricielle.
Selon la condition de stabilité, le vecteur de probabilité stationnaire est défini comme $\pi = [\pi_0, \pi_1, \pi_2, ..., \pi_\Psi]$.
En résolvant le système linéaire :

$$\begin{cases} \pi Q = 0 \\ \pi e = 1, e : vecteurcolonnedesuns. \end{cases}$$

La solution géométrique matricielle est obtenue par le système suivant

$$\begin{cases} \pi_0 = \pi_1 R_0 \\ \pi_k = \pi_1 R_k^*. \end{cases}$$

π_1 est obtenu en résolvant le système linéaire suivant.

$$\begin{cases} \pi_1 (R_0 \hat{F} + L + R_2 B) = 0 \\ \pi_1 (R_0 e_0 + \sum_{k=1}^{\Psi} R_k^* e) = 1. \end{cases}$$

Avec $R_k^* = \prod_{j=1}^{k} R_j$, R_j est la matrice de taux qui est obtenu par l'algorithme 5.7.2.

Algorithm 2: Calculer R_k

$R_\Psi \leftarrow -FL_\Psi^{-1}$
if $k > 1$ **then**
  for $j = \Psi - 1 \rightarrow k + 1$ **do**
    $R_j \leftarrow -F(L + R_{j+1}B)^{-1}$
  end for
  return $R_k \leftarrow -F(L + R_{k+1}B)^{-1}$
end if
if $k = 0$ **then**
  return $R_0 \leftarrow -\hat{B}\hat{L}^{-1}$
end if
if $k = 1$ **then**
  return $R_1 \leftarrow I$
end if

Ainsi, la probabilité stationnaire peut être calculée par l'algorithme 5.7.2

Algorithm 3: Calculate π

for $j = 1 \rightarrow k$ **do**
  $R_j \leftarrow$ **calculate R(j)**
  $\pi_j \leftarrow \pi_1 R_j$
end for

5.7.3 Résultats numériques

Dans cette section, nous évaluons le modèle d'un commutateur virtuel avec plusieurs ports d'entrée. Nous étudions l'impact de l'arrivée des lots avec une taille maximale égale à 2. Les résultats sont obtenus à l'aide de simulateur Matlab en utilisant les algorithmes 1, 5.7.2 et 5.7.2. Nous considérons que la capacité de la file d'attente RX est fixée à K = 128. Le commutateur virtuel

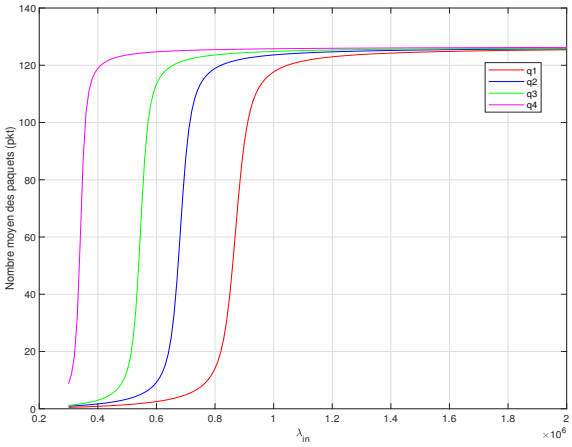


FIGURE 5.9 – Nombre moyen de paquets en fonction de λ_{in}

est composé de 4 ports d'entrée desservis par un ensemble de CPU. Nous limitons notre étude à un seul d'entre eux. Le taux de service du CPU Core est égal à 1 Mpps (millions de paquets par seconde). Pour mettre en évidence l'impact du taux d'arrivée, nous avons supposé que le taux d'entrée global des paquets, λ_{in}, est inégalement réparti entre les 4 ports comme suit : 15% sur le port 1, 20% sur le port 2, 25% sur le port 3 et 40% sur port 4.

— **Nombre moyen de paquets q_i (Eq.5.5)**
Le nombre moyen de paquets dans le système est représenté par la figure 5.9 en fonction du taux d'arrivé global λ_{in}. La figure représente 4 courbes (chaque courbe est associée à un port individuel i (i = 1,.. ., 4). Les résultats sont attendus. Chaque courbe augmente de 0 à 128 (capacité des files d'attente) et représente un comportement non linéaire. La file d'attente associée au port qui reçoit la plus petite fraction des paquets se remplit lentement. Ainsi, la taille moyenne de la file d'attente reste faible pour des valeurs plus importantes de λ_{in} (en le comparant avec la file d'attente associée au port qui reçoit trafic plus important) avant d'augmenter rapidement jusqu'à atteindre la valeur de 128. Comme prévu, la courbe violet associée au port 4, est le premier à saturer puisqu'il reçoit la plus grande fraction de paquets entrants. La courbe rouge correspondant au port 1 est le dernier à saturer, où q_1 commence à augmenter pour une valeur de λ_{in} égale à 0.6 contrairement à q_4 qui a augmenté pour une valeur de λ_{in} plus petite (0.2).

— **La probabilité de blocage b_i**
On rappelle que la probabilité de blocage est définie comme la probabilité qu'un lot arrive et trouve le système dans un état bloquant. Puisqu'on a un système de file d'attente à capacité finie, les nouveaux groupes seront rejetés s'ils arrivent alors que la file est déjà pleine ou le nombre des paquets dans un groupe est supérieur aux places disponibles. La probabilité de blocage est définie par l'équation suivante :

$$b_i = \pi(K,P) + \pi(K,R) + (\pi(K-1,P) + \pi(K-1,R)) * g_2 \tag{5.7}$$

Les probabilités de blocage, exprimées en fonction du débit d'entrée global λ_{in}, sont représentées par la figure 5.10. Elles restent proches de 0 avant d'augmenter à mesure que λ_{in} augmente. Encore une fois, parce qu'il reçoit la plus grande fraction de paquets entrants, le port 4 est le premier à subir des pertes de paquets. En comparant notre résultat avec celui de [132], nous remarquons que les probabilités de blocage gardent la même forme des courbes mais atteignent des valeurs plus petites. Cela peut s'expliquer par la stratégie d'acceptation et la discipline de service utilisée dans notre modèle.

— **La durée moyenne du séjour r_i (Eq. 5.15)**
La figure 5.11 montre les performances en termes de temps de séjour moyen par rapport au

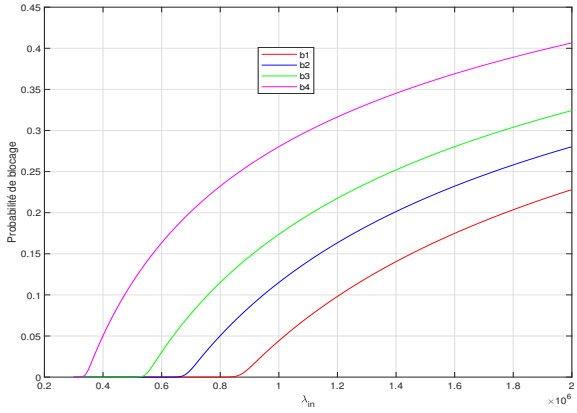


FIGURE 5.10 – La probabilité de blocage en fonction de λ_{in}

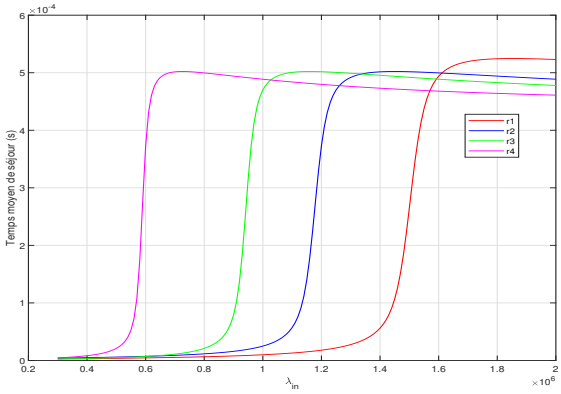


FIGURE 5.11 – La temps moyen du séjour en fonction de λ_{in}

débit d'entrée global λ_{in} . Nous considérons le temps de séjour moyen r_i passé par le paquet dans chaque file d'attente. Ce temps est composé du temps d'attente dans la file d'attente et du temps de service.
Nous rappelons que le paquet entrant peut subir trois destins : *(i)* il peut être bloqué à l'arrivée (la file est pleine), *(ii)* il peut trouver un serveur libre (il n'y a pas de paquet dans la file d'attente i et le serveur visite la file d'attente au moment de l'arrivée) et, par conséquent, se mettre directement en service, *(iii)* ou il peut rejoindre la file d'attente. Dans les deux premiers cas, il n'y a pas de temps d'attente. Dans le troisième cas, le temps d'attente dépend du nombre de paquets dans le système, alors le temps de séjour est le temps pour servir tous les paquets déjà en attente dans la file d'attente courante et tout ou partie des paquets dans les autres files d'attente. De même pour la taille moyenne de la file d'attente, la durée moyenne de séjour dans chaque port commence à augmenter à différents niveaux de charge. Il est évident que le port 4 est le premier à atteindre une valeur maximale.

5.8 Taille de groupe aléatoire

Dans cette partie, nous allons étudier le cas où les paquets arrivent en groupes de taille aléatoires. Le nombre de clients par groupe est une variable aléatoire X strictement positive avec la probabilité P(X=m) = g_m =$Pr\{$ le groupe d'arrivées contient m paquets, m ≥ 1 $\}$, avec $\sum_{m=1}^{\infty} g_m = 1$.
Nous allons analyser le système sous cette hypothèse selon les deux stratégies d'acceptation mentionnées dans la section 5.5.4. La figure 5.6 montre le diagramme de transition selon la straté-

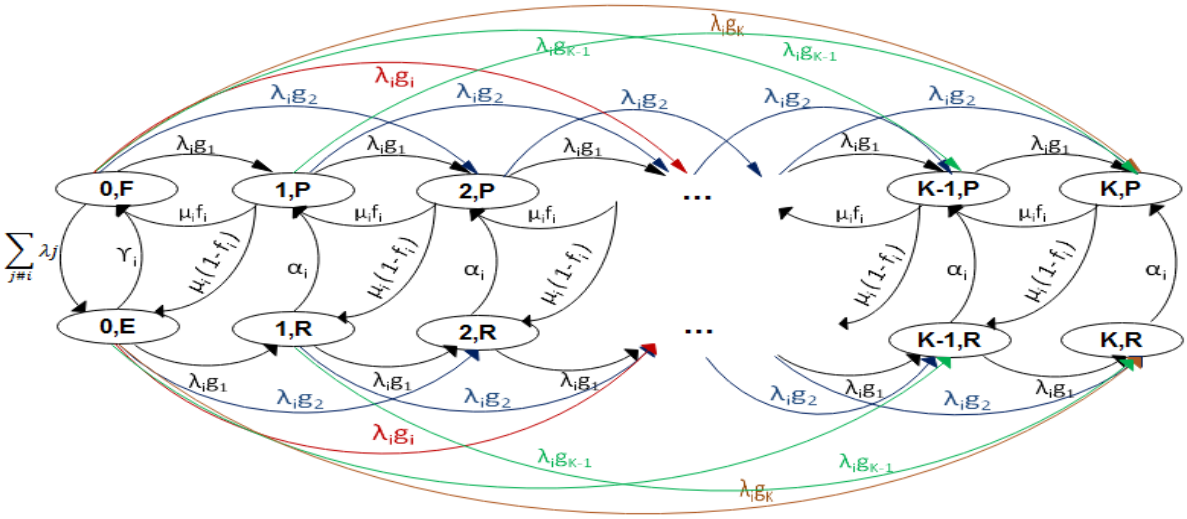


FIGURE 5.12 – Le diagramme de transition associé à la file d'attente i où la taille du groupe d'arrivées est aléatoire sous la stratégie d'acceptance totale.

gie d'acceptation totale. Le groupe d'arrivée ne sera accepté que si le nombre de paquets dans le groupe est inférieur ou égale au nombre des positions disponibles dans la file.

La figure 5.13 montre le diagramme de transition selon la stratégie d'acceptation partielle. Dans le cas où la taille du groupe est supérieure au nombre des positions libres, on remplit les positions libres et on rejette le reste.

5.8.1 Analyse

Comme le montrent les figures 5.12 et 5.13, on ne peut pas parler d'un processus QBD, car il existe des transitions entre tous les états non adjacents.
Selon le diagramme de transition, la matrice de taux de transition peut être écrit comme suit :
stratégie 1 :

$$Q_1 = \begin{pmatrix} B_0 & B_1 & B_2 & B_3 & ... & B_K \\ A_0 & A_1 & B_1 & B_2 & ... & B_{K-1} \\ 0 & A_0 & A_2 & B_1 & ... & B_{K-2} \\ 0 & 0 & A_0 & A_1 & ... & B_{K-3} \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & A_{K-1} & B_1 \\ 0 & 0 & 0 & ... & A_0 & A_K \end{pmatrix}$$

avec

$$B_0 = \begin{pmatrix} -(\sum_{i\neq j}\lambda_j + \lambda_i \sum_{n=1}^{K} g_n) & \sum_{i\neq j}\lambda_j \\ \gamma_i & -(\gamma_i + \lambda_i \sum_{n=1}^{K} g_n) \end{pmatrix},$$

$$A_0 = \begin{pmatrix} \mu_i f_i & \mu_i(1-f_i) \\ 0 & 0 \end{pmatrix}, B_n = \begin{pmatrix} \lambda_i g_n & 0 \\ 0 & \lambda_i g_n \end{pmatrix}, 1 \leq n \leq K$$

$$A_j = \begin{pmatrix} -(\mu_i + \lambda_i \sum_{n=1}^{K-i} g_n) & 0 \\ \alpha_i & -(\alpha_i + \lambda_i \sum_{n=1}^{K-i} g_n) \end{pmatrix}, 1 \leq j \leq K\text{-}1, A_K = \begin{pmatrix} -\mu_i & 0 \\ \alpha_i & -\alpha_i \end{pmatrix}$$

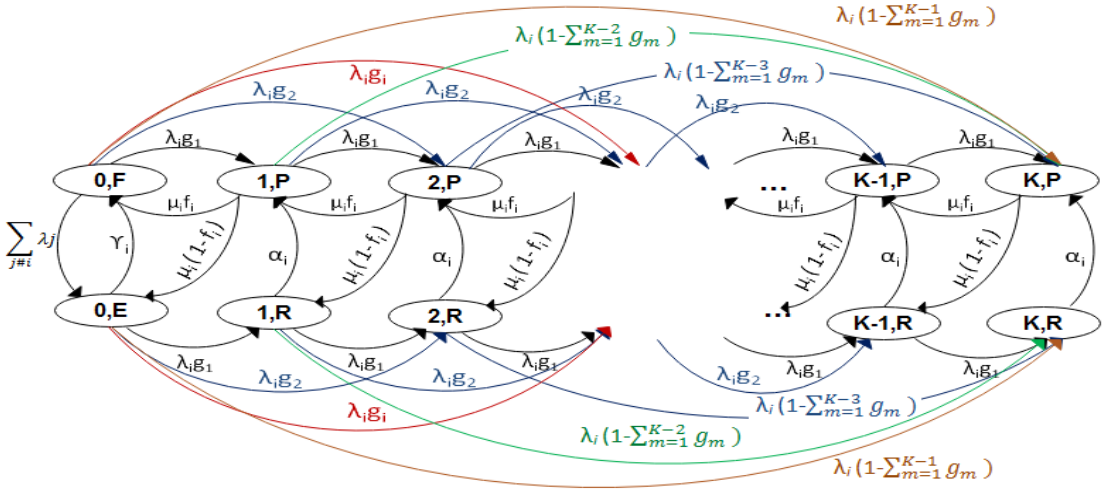


FIGURE 5.13 – Le diagramme de transition associé à la file d'attente i où la taille du groupe d'arrivées est aléatoire sous la stratégie d'acceptance partielle.

Stratégie 2 :

$$Q_2 = \begin{pmatrix} B_0 & B_1 & B_2 & B_3 & \dots & B_{K-1} & C_1 \\ A_0 & A & B_1 & B_2 & \dots & B_{K-2} & C_2 \\ 0 & A_0 & A & B_1 & \dots & B_{K-3} & C_3 \\ \dots & \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & 0 & \dots & \dots & B_1 & C_{K-1} \\ 0 & 0 & 0 & \dots & \dots & A_1 & B_1 \\ 0 & 0 & 0 & \dots & \dots & A_0 & A_K \end{pmatrix}$$

avec

$$B_0 = \begin{pmatrix} -(\sum_{i\neq j} \lambda_j + \lambda_i) & \sum_{i\neq j} \lambda_j \\ \gamma_i & -(\gamma_i + \lambda_i) \end{pmatrix}, A_0 = \begin{pmatrix} \mu_i f_i & \mu_i(1-f_i) \\ 0 & 0 \end{pmatrix}, B_n = \begin{pmatrix} \lambda_i g_n & 0 \\ 0 & \lambda_i g_n \end{pmatrix}, 1 \leq n \leq K\text{-}1$$

$$C_j = \begin{pmatrix} \lambda_i(1 - \sum_{n=1}^{K-j} g_n & 0 \\ 0 & \lambda_i(1 - \sum_{n=1}^{K-j} g_n \end{pmatrix}, 1 \leq j \leq K\text{-}1$$

$$A = \begin{pmatrix} -(\mu_i + \lambda_i) & 0 \\ \alpha_i & -(\alpha_i + \lambda_i) \end{pmatrix}, A_1 = \begin{pmatrix} -(\mu_i + \lambda_i g_1) & 0 \\ \alpha_i & -(\alpha + \lambda_i g_1) \end{pmatrix}, A_K = \begin{pmatrix} -\mu_i & 0 \\ \alpha_i & -\alpha_i \end{pmatrix}$$

Sous la condition de stabilité, le vecteur de probabilité stationnaire est défini comme $\pi = [\pi_0,\pi_1,\pi_2,\dots,\pi_K]$. Il peut être trouvée en résolvant $\pi.Q = 0$ avec l'équation de normalisation $\pi.e = 1$, où e est un vecteur colonne d'unité. Supposons que $\pi_0=(\pi(0,F),\pi(0,E))$ et $\pi_i=(\pi(i,P),\pi(i,R))$ where $1 \leq i \leq K$

5.8.2 Probabilités de blocage

En considérant deux stratégies d'acceptation, nous avons distingué entre la probabilité de blocage pour un lot arbitraire et la probabilité de blocage d'un paquet arbitraire. Tout état peut être un état bloquant puisque nous avons un espace d'attente de capacité finie et un lot de taille aléatoire.

— La probabilité de blocage d'un lot est la probabilité que le lot soit partiellement ou totalement rejeté. D'après [147], son expression est la même pour chaque stratégie d'acceptation et donnée comme suit.

$$B_{batch} = \sum_{k=0}^{K} \pi_k \sum_{j=K-k+1}^{\infty} g_j \tag{5.8}$$

Si le lot trouve k paquets dans le système, le lot sera bloqué si sa taille est supérieure à K-k.

Il est vrai que l'expression de la probabilité de blocage d'un lot 5.8 est indépendante de la stratégie d'acceptation, cependant, elle dépendra de la stratégie puisque les deux stratégies ont des probabilités d'état différentes.

— La probabilité de blocage d'un paquet arbitraire : D'après [148], le paquet arbitraire occupe n'importe quelle position dans son lot avec une probabilité égale à

$$\frac{jg_j}{\text{MBS}} \tag{5.9}$$

MBS représente la taille moyenne du lot.
Ainsi, les expressions résultantes de la probabilité de blocage d'un paquet arbitraire sont données par Eq. 5.10 pour *stratégie 1* et Eq. 5.11 pour *stratégie 2* selon [147].

Strategy 1 :

$$\text{B}_{packet} = \frac{1}{\text{MBS}} \sum_{k=0}^{\text{K}} \pi_k \sum_{j=\text{K}-k+1}^{\infty} jg_j \tag{5.10}$$

Selon l'équation 5.10 le lot contenant le paquet arbitraire est plus grand que l'espace d'attente disponible.
Strategy 2 :

$$\text{B}_{packet} = \frac{1}{\text{MBS}} \sum_{k=0}^{\text{K}} \pi_k \sum_{j=\text{K}-k+1}^{\infty} (j-\text{K}+k)g_j \tag{5.11}$$

Dans la *stratégie 2*, lorsque le lot est plus grand que l'espace disponible, le paquet arbitraire occupe une position dans la partie rejetée du lot avec probabilité (j-K+k)/j. Les équations 5.10 et 5.11 peuvent être interprétées comme rapport entre le nombre de paquets perdus et le nombre de paquets arrivant sur une longue période. En effet, les numérateurs de 5.10 et 5.11) sont considérés comme les longueurs attendues des lots bloqués (ou des lots partiellement bloqués), et les dénominateurs comme les longueurs attendues de tous les lots.

5.8.3 Résultats numériques

Dans cette section, nous allons évaluer notre modèle de file d'attente de commutateur virtuel avec différents paramètres. Nous étudions l'impact de l'arrivée des lots sur les métriques de performances selon deux stratégies d'acceptation proposées *stratégie 1* et *stratégie 2*. Les résultats sont obtenus en utilisant le simulateur Matlab.
On rappelle que le nombre de paquets X qui arrivent au service en même temps est une variable aléatoire dont la distribution est donnée par P(X=m) = g_m, (m= nombre des paquets dans le groupe) avec une moyenne E(X)= $\overline{g}$. La fonction de probabilité g_m satisfait la distribution géométrique et elle a la forme :

$$g_m=\theta^{m-1}(1\text{-}\theta),\ i\geq 1 \text{ and } 0<\theta<1\,;$$

avec $\overline{g}$ la taille moyenne du lot (MBS), donné par

$$\text{MBS} = \frac{1}{1-\theta} \tag{5.12}$$

où θ est le paramètre de la distribution géométrique..
Sous la condition de stabilité, le vecteur de probabilité stationnaire est défini comme $\pi = [\pi_0,\pi_1,\pi_2,...,\pi_K]$ peut être trouvé en résolvant π.Q =0 avec l'équation de normalisation π.e = 1, où e est un vecteur-colonne des uns.
On suppose que π_0=(π(0,F),π(0,E)) et π_i=(π(i,P),π(i,R)) avec $1 \leq i \leq K$

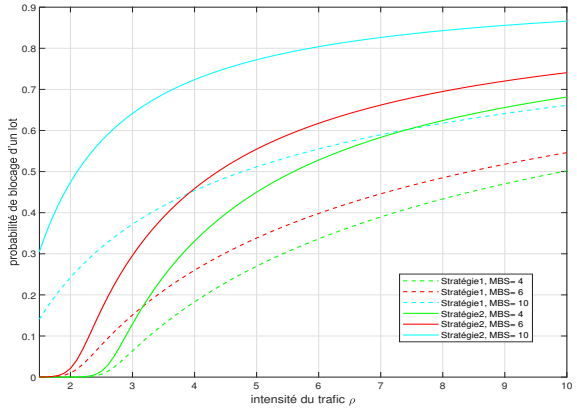


FIGURE 5.14 – La probabilité de blocage d'un lot en fonction de l'intensité de trafic ρ

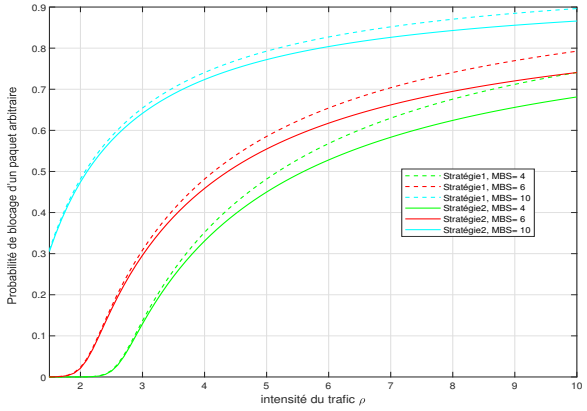


FIGURE 5.15 – La probabilité de blocage d'un paquet en fonction de l'intensité de trafic vs ρ

5.8.3.1 Paramètres de performances

Dans cette section, nous évaluons l'effet de la taille moyenne de lot et de la stratégie de l'acceptance sur les paramètres de performances tels que la longueur moyenne de la file d'attente, le temps moyen de séjour dans le système et la probabilité de blocage. Nous limitons notre étude à un seul CPU qui dessert 4 files d'attente RX. Supposons que la capacité de la file d'attente RX est égale à K = 128 et que le débit d'entrée global des paquets est également réparti entre les 4 ports.

— La probabilité de blocage par lots et la probabilité de blocage d'un paquet arbitraire sont présentées sur les figures 5.14 et 5.15 en fonction de l'intensité du trafic qui est donné par l'équation

$$\rho = \frac{\lambda \text{MBS}}{\mu} \tag{5.13}$$

La probabilité de blocage augmente de 0 à 1 à mesure que l'intensité du trafic augmente. Il est clair que les probabilités de blocage sont sensibles à la taille du lot (MBS=4, 6, 10), de sorte que les caractéristiques du trafic d'entrée doivent être soigneusement modélisées.

La figure 5.14 montre que la stratégie 2 «acceptation partielle du lot» présente une probabilité plus élevée, que la stratégie 1 «acceptation totale du lot» . Ceci peut s'expliquer par le fait que le nombre de paquets présents dans la file d'attente sous la stratégie 2 est plus grand que celui sous la stratégie 1, puisque avec cette stratégie on remplit toujours les positions disponibles donc la file est pleine la plupart du temps.

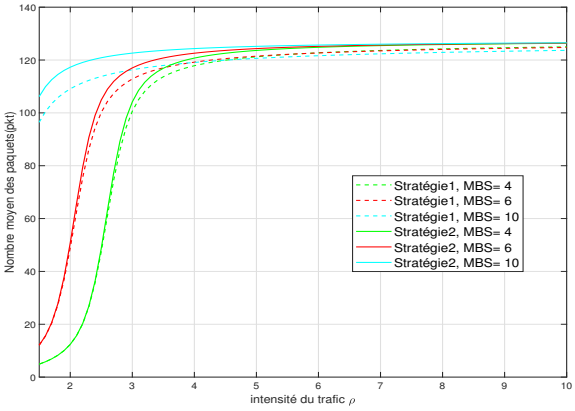


FIGURE 5.16 – Le nombre moyen des paquets en fonction de l'intensité de trafic ρ

D'autre part, c'est le cas opposé présent pour la probabilité de blocage d'un paquet arbitraire où la figure 5.15 montre que la stratégie 1 permet une probabilité de blocage élevée, ceci est expliqué par l'expression de Eq. 5.10 et Eq. 5.11..

— La longueur moyenne de la file d'attente en fonction de l'intensité du trafic est donnée par la figure 5.16 et donnée par l'équation 5.14.

$$q_i = \sum_{k=1}^{K} k.\pi_k \tag{5.14}$$

Toutes les courbes augmentent de 0 à 128 (capacité du système) et affichent un comportement non linéaire. Avec une augmentation de la taille moyenne des lots, une croissance du nombre moyen de paquets trouvés dans le système se produit. Il est clair que pour un groupe de taille moyenne contenant plus des paquets, la file d'attente sature rapidement. Sous la stratégie 2, nous avons un plus grand nombre de paquets que la stratégie 1. C'est évident puisque avec la stratégie 2 on remplit toujours les places disponibles.

— La durée moyenne du séjour en fonction de l'intensité du trafic est représentée par la figure 5.17. Cette durée est obtenue en utilisant la loi de Little

$$r_i = \frac{q_i}{\lambda_i^*} \tag{5.15}$$

avec λ_i^* est le taux d'arrivée effectif du paquet, donné par ce qui suit :
Stratégie 1

$$\lambda_i^* = \lambda_i \sum_{k=0}^{K} \pi_k \sum_{j=0}^{K-k} j g_j \tag{5.16}$$

Stratégie 2

$$\lambda_i^* = \lambda_i \sum_{k=0}^{K} \pi_k [\sum_{j=0}^{K-k} j g_j + \sum_{j=K-k+1}^{\infty} (K-k) g_j] \tag{5.17}$$

Le temps de séjour moyen augmente en fonction de l'intensité du trafic pour les deux stratégies comme prévu, mais un effet intéressant se produit en fonction de la taille moyenne des lots. À faible charge, il y a peu de blocage et l'arrivée des groupes à grande taille augmente le temps d'attente dans la file d'où un temps des séjour plus important. Cependant avec un trafic élevé, la probabilité de blocage augmente pour les lots volumineux signifie que relativement moins de clients sont acceptés pour ces lots et donc moins de temps de séjour dans la file.

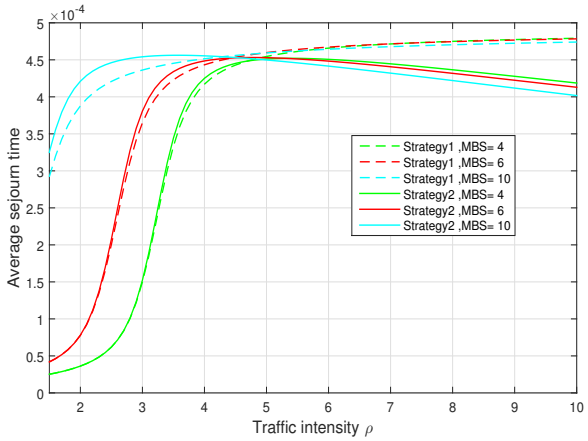


FIGURE 5.17 – Le temps moyen de séjour en fonction de l'intensité de trafic ρ

Les résultats montrent clairement que la taille du groupe du processus d'arrivée est un facteur important qui affecte les performances du système. De plus, la stratégie d'acceptation des groupes adoptée par un système a une influence significative sur les performances. Par conséquent, selon l'application du réseau à commutation de paquets, on peut choisir une stratégie. Par exemple, pour des applications où il est important que tous les paquets d'un message doivent rester ensemble, la stratégie 1 doit être utilisée.
Enfin, l'analyse de ce modèle peut être utilisée pour une classe plus large de problèmes utilisant des processus d'arrivée de lots.

5.9 Conclusion

Dans ce chapitre, nous sommes intéressés par la virtualisation des fonctions de réseau. Nous avons basé sur le modèle analytique de file d'attente proposé par [132] qui évalue les performances d'un commutateur virtuel avec plusieurs cartes d'interface réseau et plusieurs cœurs de processeur. Un système de polling a été proposé, dans lesquels un ensemble de serveurs (CPU) interroge séquentiellement des paquets à partir d'un ensemble de files d'attente afin d'évaluer le comportement du commutateur. Cependant, pour éviter la croissance de l'espace d'états associé à ces modèles, l'approche proposée découple le système d'interrogation associé à chaque CPU en plusieurs files d'attente, et recourt à des serveurs avec vacances pour capturer les interactions entre les files d'attente. Notre contribution est d'étudier le cas où les paquets arrivent par groupes. On a étudié deux cas des groupes d'arrivée de taille fixe ou taille aléatoire. Puisque la file d'attente a une capacité finie, nous avons évalué les métriques de performances telles que la probabilité de blocage et le temps de séjour sous deux stratégies d'acceptation (acceptation totale d'un lot et acceptation partielle d'un lot).

Chapitre 6

Conclusion et perspectives

6.1 Rappel du contexte

Aujourd'hui, les entreprises s'appuient de plus en plus sur le cloud computing (l'informatique "dans les nuages") qui met à leur disposition différents services (tel que hébergement des services, stockage de données) ou des infrastructures avec des grandes capacités de calculs et de stockages. Ces services sont facturés à la demande, les utilisateurs ne payent que les services et les ressources utilisées. Trois niveaux de service sur lesquels repose le Cloud Computing : Infrastructure as a Service (IaaS), Plateform as a Service (PaaS) et Software as a Service (SaaS). Au niveau Iaas, les fournisseurs de Cloud mettent à disposition une infrastructure complète sous forme de machines virtuelles (VM), des connexions et des composants réseau tels que des commutateurs, des routeurs et des pare-feu. La mise en place de cette infrastructure est assurée grâce aux techniques de virtualisation permettant d'utiliser efficacement les ressources des serveurs (processeur, mémoire, E/S réseau, etc).
Aussi, les fournisseurs de cloud s'engagent à garantir le bon fonctionnement des ressources ainsi que leurs disponibilités. Pour faire face aux fluctuations de la charge de travail et à l'évolution de la demande des utilisateurs, des centres de données virtualisés ont été déployés permettant de prendre en charge des applications à grande échelle et de stocker de gros volumes de données. Cependant, ces centres souffrent d'un certain nombre de limitations telles que les performances des ressources et leurs disponibilités (par exemple ; les machines virtuelles qui sont sujettes à plusieurs pannes logiciels qui peuvent affecter la qualité de service prévisible de certaines applications), la sécurité du réseau, la maîtrise de la consommation d'énergie....
Le long de ce travail nous sommes intéressés par l'infrastructure IaaS, plus précisément, la virtualisation des serveurs et celle du réseau (nous avons étudié l'un des composants clé du réseau le commutateur virtuel). Nous résumons les contributions de cette thèse dans la section suivante. Quelques perspectives de futures recherches sont ensuite proposées à la fin de ce chapitre.

6.2 Résumé des contributions

Dans les travaux réalisés nous avons focalisé sur le niveau de services Iaas sur lequel porte les contributions de cette thèse. Nous sommes intéressés par la modélisation qui nous permet d'évaluer des différents métriques de performances sous différentes hypothèses. Dans un premier lieu, nous avons étudié la sûreté de fonctionnement des systèmes virtualisés avec une maîtrise de la consommation de l'énergie, en utilisant SRN comme formalismes de modélisation. Dans un deuxième lieu, nous avons étudié les performances du commutateur virtuel en utilisant la théorie des files d'attente sous l'hypothèse des arrivées des paquets groupées.
La première partie de ce travail présente le contexte de cette thèse. Dans un premier temps, nous avons introduit les thèmes du Cloud Computing, la virtualisation des serveurs et du réseau, ainsi nous avons présenté quelques défis rencontrés comme le vieillissement logiciel et le gaspillage

énergétique. Dans un second temps, nous avons focalisé sur la modélisation stochastique pour l'analyse des performances des systèmes où nous avons présenté les différents formalismes de modélisation ainsi que les méthodes analytiques et numériques utilisées pour résoudre et analyser les différents modèles.
La deuxième partie de cette thèse présente les différentes contributions apportées :

— **Performabilité et gestion de l'énergie pour les SVSs**
Dans le chapitre 4, nous nous sommes intéressés à l'étude de la performabilité des SVSs déployant des mécanismes de gestion de l' énergie exploitant les fonctionnalités ACPI. Notre défi est de trouver un bon compromis puissance-performance permettant de respecter les contraintes des applications supportées et maîtriser en même temps la dépense énergétique. Cette problématique a été traitée au travers le développement de modèles de disponibilité basés SRNs capturant plusieurs caractéristiques des SVSs. Nous avons mis en évidence le comportement d'un SVS composé d'un VMM, d'un VM et d'un PMC supportant des trafics sporadiques. Nous avons considéré que le VMM et le VM dans le modèle SVS sont soumis aux aléas du vieillissement logiciel qui cause une dégradation des performances ou un taux d'échec croissant des machines. Afin de lutter contre ce phénomène, le rajeunissement logiciel a été utilisé comme une méthode proactive dans le but de prévenir les pannes. Nous avons étudié deux techniques de rajeunissement VMM qui sont le rajeunissement Cold-VM et Migrate-VM.
De plus, le modèle SVS intègre un mécanisme de gestion de l'énergie basé sur le temps en adaptant les états d'énergie du PMC aux variations du trafic.
L'exploitation du modèle a été faite au travers la définition de plusieurs métriques mettant en évidence les aspects de performance; performabilité et d'efficacité énergétique. Nous avons étudié l'impact de la charge de travail et de l'indice de dispersion sur les métriques de performance.

— **Modélisation des performances du commutateur virtuel**
Le chapitre 5 a été consacré pour la deuxième contribution. Nous nous sommes orientés vers la virtualisation des réseaux et ses composants. L'élément réseau étudié dans cette thèse est le commutateur virtuel. Ce dernier offre un niveau élevé de flexibilité dans la gestion de ses ressources telles que le nombre de cœurs de processeur (CPU), le nombre des ports, et les capacités des files d'attente RX, ce qui donne la possibilité d'un dimensionnement efficace des ressources système. Nous nous sommes basé sur le modèle proposé par les auteurs dans [132] qui ont proposé un modèle pour l'évaluation des performances d'un commutateur virtuel basé sur les files d'attente avec des serveurs avec vacances. Les paquets arrivent indépendamment selon un processus de Poisson. Cependant, cette hypothèse n'est plus réaliste, c'est pour cela, nous avons étendu cette hypothèse en considérant que les paquets arrivent au file d'attente selon le processus de Poisson par groupe. Dans un premier temps, nous avons étudié le cas où la taille du groupe fixe. Nous avons montré sous certaines hypothèses que la matrice génératrice infinitésimale a une forme quasi-naissance-mort (QBD) qui a simplifié l'évaluation analytique des performances. Dans un deuxième temps, nous avons évalué le modèle sous l'hypothèse d'une taille du groupe aléatoire. Nous avons proposé une file d'attente MX/M/1/K avec une capacité finie. Les métriques de performances telles que les probabilités de blocage et le temps de séjour des paquets sont obtenues selon deux stratégies d'acceptation des lots.
Les résultats numériques seront utiles pour le dimensionnement des ressources d'un commutateur virtuel afin de faire face à la variation de la charge de travail.

6.3 Perspectives

Notre travail sur la virtualisation dans les centres de données virtualisés peut être potentiellement étendu dans un certain nombre de directions. Dans ce qui suit, nous allons discuter de

certains des plus importants :

— Le rajeunissement logiciel est effectué après chaque période et en fonction de la charge de travail. On peut aussi mettre en place un système d'analyse et de gestion d'infrastructure qui collecte et surveille les métriques statiques des VMM et des VM (l'utilisation du CPU, l'utilisation de la mémoire, l'accès au disque,...). Un seuil peut être défini, et une fois atteint l'action de rajeunissement sera déclenchée afin d'éviter un crash ou un accident du système.

— Évaluer le modèle de SVS avec un hyperviseur comportant plusieurs VM. Dans tel cas, on doit mettre en place des algorithmes de migration permettant de choisir la meilleure destination et assurer un équilibrage de charge afin d'assurer une bonne performance de VMM.

— On a évalué les performances du commutateur virtuel en supposant que le CPU traite un seul paquet à chaque visite. Cette hypothèse peut être étendue où on suppose que les paquets seront servis par groupe. Dans ce cas on peut mettre en place un système de sélection des paquets qui nécessitent un traitement ou des ressources identiques.

— Utiliser des plateformes de simulation afin de confirmer nos résultats trouvées avec la modélisation.

— Dans cette thèse, nous nous sommes intéressés par la virtualisation de serveurs et des fonctions réseau, on peut intégrer la virtualisation de stockage qui représente un élément important dans les centres de données.

Bibliographie

[1] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing : Vision, hype, and reality for delivering it services as computing utilities. In *2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 5–13, 2008. doi : 10.1109/HPCC.2008.172. 9, 12

[2] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds : Towards a cloud definition. 2009. 9

[3] Jeremy Geelan. Twenty one experts define cloud computing. *Virtualization*, 08 2008. 9

[4] Barrie Sosinsky. *Cloud computing bible*, volume 762. John Wiley & Sons, 2010. ISBN 9781118255674. doi : 10.1002/9781118255674. 9

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4) :50–58, April 2010. ISSN 0001-0782. doi : 10.1145/1721654.1721672. URL https://doi.org/10.1145/1721654.1721672. 9

[6] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011. ISBN 9780470887998. 9

[7] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, USA, 2011. 9

[8] Amazon Elastic. Amazon elastic compute cloud (amazon ec2), 2010. URL https://olivierleclere.ch/blog/2010/01/amazon-elastic-compute-cloud-amazon-ec2/. 9

[9] Krishnan Saidapet P. T. and booktitle = In Building Your Next Big Thing with Google Cloud Platform chapter = 5 pages = 53–81 publisher = Apress, Berkeley, CA year = 2015 = https ://doi.org/10.1007/978-1-4842-1004-8-4 Jose L. Ugia Gonzalez, title = Google compute engine . 9

[10] Rahul Ghosh, Francesco Longo, Vijay Naik, and Kishor Trivedi. Modeling and performance analysis of large scale iaas clouds. *Future Generation Computer Systems*, 29 :1216 – 1234, 07 2013. doi : 10.1016/j.future.2012.06.005. 9

[11] Google. Google cloud platform, 2019. URL https://cloud.google.com/. 9

[12] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack : Toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55 : 38–42, 10 2012. doi : 10.5120/8738-2991. 10

[13] Rakesh Kumar, Kanishk Jain, Hitesh Maharwal, Neha Jain, and Anjali Dadhich. 07 2014. 10

[14] Rafael Moreno-Vozmediano, Rubén Montero, and Ignacio Llorente. Iaas cloud architecture : From virtualized datacenters to federated cloud infrastructures. *Computer*, 45 :65–72, 12 2012. doi : 10.1109/MC.2012.76. 10

[15] CloudStack. Apache cloudstack : Open source cloud computing, 2019. URL https://cloudstack.apache.org/. 10

[16] Alexander Zahariev. Google app engine. In *TKK T-110.5190 Seminar on Internetworking*, 2009. 10

[17] Microsoft. Microsoft azure, 2019. URL https://azure.microsoft.com/. 10

[18] Nancy Conner. *Google Apps : The Missing Manual*. O'Reilly Media, Inc., 2008. ISBN ISBN : 9780596515799. 10

[19] Susanta Nanda and Tzi cker Chiueh. A survey on virtualization. In *RPE Report*, page 1–42. 2008. 12

[20] Robert P Goldberg. Architecture of virtual machines. In *Proceedings of the Workshop on Virtual Computer Systems*, page 74–112, New York, NY, USA, 1973. Association for Computing Machinery. ISBN 9781450374279. doi : 10.1145/800122.803950. URL https://doi.org/10.1145/800122.803950. 12

[21] Barham Paul, Dragovic Boris, Fraser Keir, Hand Steven, Harris Tim, Ho Alex, Neugebauer Rolf, Pratt Ian, and Warfield Andrew. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, page 164–177, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137575. doi : 10.1145/945445.945462. URL https://doi.org/10.1145/945445.945462. 12

[22] VMware. Understanding full virtualization, paravirtualization, and hardware assist, vmware white paper, 2007. 12

[23] VMware ESXi. URL http://www.vmware.com/products/vsphere/esxiandesx/index.html. 12

[24] Leinenbach Dirk and Santen Thomas. Verifying the microsoft hyper-v hypervisor with vcc. In *Proceedings of the 2nd World Congress on Formal Methods*, page 806–809. Springer-Verlag, 2009. ISBN 9783642050886. doi : 10.1007/978-3-642-05089-3_51. URL https://doi.org/10.1007/978-3-642-05089-3_51. 12

[25] Microsoft Hyper-V. URL http://www.microsoft.com/hyper-v-server/. 12

[26] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. Kvm : the linux virtual machine monitor. In *In Proceedings of the 2007 Ottawa Linux Symposium (OLS-07*, 2007. 12, 21

[27] Menascé Daniel A. Virtualization : Concepts, applications, and performance modeling. In *Int. CMG Conference*, pages 407–414. Computer Measurement Group, 2005. URL http://dblp.uni-trier.de/db/conf/cmg/cmg2005.html#Menasce05. 12, 21

[28] Huang Wei, Liu Jiuxing, Abali Bulent, and Panda Dhabaleswar K. A case for high performance computing with virtual machines. In *Proceedings of the 20th Annual International Conference on Supercomputing*, ICS 06, page 125–134. Association for Computing Machinery, 2006. ISBN 1595932828. doi : 10.1145/1183401.1183421. URL https://doi.org/10.1145/1183401.1183421. 13

[29] Gerald J Popek and Robert P.Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7) :412–421, 1974. ISSN 0001-0782. doi : 10.1145/361011.361073. URL https://doi.org/10.1145/361011.361073. 13

[30] Al Muller and Seburn Wilson. *Virtualization with vmware esx server*. The name of the publisher, 2005. ISBN 978-1-59749-019-1. 13

[31] Velte Anthony and Velte Toby. *Microsoft Virtualization with Hyper-V*. McGraw-Hill, Inc., USA, 1 edition, 2009. ISBN 0071614036. 13

[32] Barham Paul, Dragovic Boris, Fraser Keir, Hand Steven, Harris Tim, Ho Alex, Neugebauer Rolf, Pratt Ian, and Warfield Andrew. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, page 164–177, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137575. doi : 10.1145/945445.945462. URL https://doi.org/10.1145/945445.945462. 13

[33] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. Kvm : the linux virtual machine monitor. In *In Proceedings of the 2007 Ottawa Linux Symposium (OLS'-07*, pages 225–230. 13

[34] Vmware workstation / player, 2019. URL .https://www.vmware.com/fr/products/workstation./. 13

[35] Microsoft virtual pc, 2019. URL https://www.microsoft.com/fr-FR/download/details.aspx?id=3702/. 13

[36] Oracle virtualbox, 2019. URL https://www.virtualbox.org/. 13

[37] Yennun Huang, Chandra Kintala, N. Kolettis, and N. D. Fulton. Software rejuvenation : analysis, module and applications. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 381–390, 1995. 16

[38] Michael Grottke, Matias Jr. Rivalino, and Kishor S. Trivedi. The fundamentals of software aging. In *2008 IEEE International Conference on Software Reliability Engineering Workshops (ISSRE Wksp)*, pages 1–6, 2008. 16

[39] Cotroneo Domenico, Natella Roberto, Pietrantuono Roberto, and Russo Stefano. A survey of software aging and rejuvenation studies. *J. Emerg. Technol. Comput. Syst.*, 10(1), January 2014. ISSN 1550-4832. doi : 10.1145/2539117. URL https://doi.org/10.1145/2539117. 16

[40] Eliot Marshall. Fatal error : How patriot overlooked a scud. *Science (New York, N.Y.)*, 255 : 1347, 04 1992. doi : 10.1126/science.255.5050.1347. 16

[41] L. Bernstein and Chandra Kintala. Software rejuvenation. *CrossTalk*, 6 :23–26, 08 2004. 16

[42] Yennun Huang, Chandra Kintala, N Kolettis, and N Fulton. Software rejuvenation : analysis, module and applications. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 381–390, 1995. doi : 10.1109/FTCS.1995.466961. 16

[43] ELIOT MARSHALL. Fatal error : How patriot overlooked a scud. *Science*, 255(5050) :1347–1347, 1992. ISSN 0036-8075. doi : 10.1126/science.255.5050.1347. URL https://science.sciencemag.org/content/255/5050/1347. 16

[44] Lawrence Bernstein. Innovative technologies for preventing network outages. *AT T Technical Journal*, 72(4) :4–10, 1993. doi : 10.1002/j.1538-7305.1993.tb00545.x. 16

[45] S. Garg, Aad van Moorsel, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi. A methodology for detection and estimation of software aging. In *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257)*, pages 283–292, 1998. 16

[46] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. Software aging analysis of the linux operating system. In *2010 IEEE 21st International Symposium on Software Reliability Engineering*, pages 71–80, 2010. 16

[47] Michael Grottke, Lei Liz, Kalyan Vaidyanathan, and Kishor S. Trivedi. Analysis of software aging in a web server. *Friedrich-Alexander-University Erlangen-Nuremberg, Chair of Statistics and Econometrics, Discussion Papers*, 55, 11 2005. doi : 10.1109/TR.2006.879609. 16

[48] Trivedi Kishor S. and Vaidyanathan Kalyanaraman. *Software Aging and Rejuvenation*. American Cancer Society, 2007. ISBN 9780470050118. doi : 10.1002/9780470050118.ecse394. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470050118.ecse394. 16

[49] Autran Macêdo, Tais B. Ferreira, and Rivalino Matias. The mechanics of memory-related software aging. In *2010 IEEE Second International Workshop on Software Aging and Rejuvenation*, pages 1–5, 2010. 16

[50] 16

[51] Yi-Min Wang, Yennun Huang, Kiem-Phong Vo, Pe-Yu Chung, and C. Kintala. Checkpointing and its applications. *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 22–31, 1995. 16

[52] BenjaminWah. *Software Aging and Rejuvenation*. 2008. 17

[53] Fumio Machida, Jianwen Xiang, Kumiko Tadano, and Yoshiharu Maeno. Combined server rejuvenation in a virtualized data center. In *9th IEEE International Conference on Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC)*, pages 486–493, 09 2012. ISBN 978-1-4673-3084-8. doi : 10.1109/UIC-ATC.2012.52. 17, 44

[54] Avi Qumranet, Yaniv Qumranet, Dor Qumranet, Uri Qumranet, and Anthony Liguori. Kvm : The linux virtual machine monitor. *Proceedings Linux Symposium*, 15, 01 2007. 17

[55] Kenichi Kourai and Shigeru Chiba. Fast software rejuvenation of virtual machine monitors. *Dependable and Secure Computing, IEEE Transactions on*, 8 :839 – 851, 01 2012. doi : 10.1109/TDSC.2010.20. 18

[56] Kenichi Kourai. A fast rejuvenation technique for server consolidation with virtual machines. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, page 245–255, 2007. 18

[57] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12) :33–37, 2007. doi : 10.1109/MC.2007.443. 19

[58] Stephen Dawson-Haggerty, Andrew Krioukov, and David E. Culler. Power optimization – a reality check. Technical Report UCB/EECS-2009-140, EECS Department, University of California, Berkeley, Oct 2009. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-140.html. 19

[59] Khiem Mike Ebbers Alvin Galea ichael Schaefer, Marc Tu Duy. The green datacenter : Steps for the journey. Technical report, IBM, 2009. 19

[60] Gregor von Laszewski, Lizhe Wang, Andrew J. Younge, and Xi He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, 2009. doi : 10.1109/CLUSTR.2009.5289182. 19

[61] Kyong Hoon Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of virtual machines for real-time cloud. *Concurrency and Computation : Practice and Experience*, 23 :1491–1505, 09 2011. doi : 10.1002/cpe.1712. 19

[62] Nathan Whittington, Lu Liu, Bo Yuan, and Marcello Trovati. Investigation of energy efficiency on cloud computing. In *2015 IEEE International Conference on Computer and Information Technology ; Ubiquitous Computing and Communications ; Dependable, Autonomic and Secure Computing ; Pervasive Intelligence and Computing*, pages 2080–2087, 2015. doi : 10.1109/CIT/IUCC/DASC/PICOM.2015.309. 20

[63] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2) :114–119, February 2013. ISSN 0163-6804. doi : 10.1109/MCOM.2013.6461195. 21

[64] Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for sdn ? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7) : 36–43, July 2013. ISSN 0163-6804. doi : 10.1109/MCOM.2013.6553676. 21

[65] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking : A comprehensive survey. *Proceedings of the IEEE*, 103(1) :14–76, Jan 2015. ISSN 0018-9219. doi : 10.1109/JPROC.2014.2371999. 21

[66] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow : Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2) :69–74, March 2008. ISSN 0146-4833. 21

[67] Raj Jain and Subharthi Paul. Network virtualization and software defined networking for cloud computing : a survey. *IEEE Communications Magazine*, 51(11) :24–31, November 2013. ISSN 0163-6804. doi : 10.1109/MCOM.2013.6658648. 21

[68] Ahmet Cihat Baktir, Atay Ozgovde, and Cem Ersoy. How can edge computing benefit from software-defined networking : A survey, use cases, and future directions. *IEEE Communications Surveys Tutorials*, 19(4) :2359–2391, Fourthquarter 2017. ISSN 1553-877X. doi : 10.1109/COMST.2017.2717482. 21

[69] Samaresh Bera, Sudip Misra, and Athanasios V. Vasilakos. Software-defined networking for internet of things : A survey. *IEEE Internet of Things Journal*, 4(6) :1994–2008, Dec 2017. ISSN 2327-4662. doi : 10.1109/JIOT.2017.2746186. 21

[70] A. Lee, H. Nguyen X. Wang, and I.Ra. A hybrid software defined networking architecture for next-generation iots. *KSII Transactions on Internet and Information Systems*, 12(2) :932–945, 2018. doi : 10.3837/tiis.2018.02.024. 21

[71] Elias Molina and Eduardo Jacob. 21

[72] Manisha Chahal, Sandeep Harit, Krishn K. Mishra, Arun Kumar Sangaiah, and Zhigao Zheng. A survey on software-defined networking in vehicular ad hoc networks : Challenges, applications and use cases. *Sustainable Cities and Society*, 35, 07 2017. doi : 10.1016/j.scs.2017.07.007. 21

[73] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem : Network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 13–24, 2012. ISBN 978-1-4503-1419-0. 22

[74] NFV White Paper. Network functions virtualisation (nfv) ; architectural framework. September 2014. 22

[75] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization : State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, 18(1) :236–262, Firstquarter 2016. ISSN 1553-877X. doi : 10.1109/COMST.2015.2477041. 22

[76] C. Cui. Network functions virtualisation : Network operator perspectives on industry progress. *SDN & OpenFlow World Congress", Dusseldorf-Germany.*, 3(1), October 2014. 22

[77] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 22

[78] Daniel M. Batista, Gordon Blair, Fabio Kon, Raouf Boutaba, David Hutchison, Raj Jain, Ramachandran Ramjee, and Christian Esteve Rothenberg. Perspectives on software-defined networks : interviews with five leading scientists from the networking community. *Journal of Internet Services and Applications*, 6(1) :22, Oct 2015. ISSN 1869-0238. doi : 10.1186/s13174-015-0035-3. URL https://doi.org/10.1186/s13174-015-0035-3. 22

[79] Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962. 28

[80] Forest Baskett, K. Mani Chandy, Richard R. Muntz, and Fernando G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22 :248–260, 1975. 31

[81] Kleinrock Leonard. *Queueing Systems-Theory*, volume 1&2. Chichester : John Wiley & Sons, 1975. 31

[82] Martin. Reiser and Stephen S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *J. ACM*, 27(2) :313–322, April 1980. ISSN 0004-5411. doi : 10.1145/322186.322195. URL https://doi.org/10.1145/322186.322195. 31

[83] William J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994. 31

[84] Youssef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003. ISBN 0898715342. 31, 32

[85] Patrick Lascaux and Raymond Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur*, volume 1,2. Paris : Masson, New York, NY, USA, 1994. 32

[86] Victor Lew Wallace. *The Solution of Quasi Birth and Death Processes Arising from Multiple Access Computer Systems*. PhD thesis, Ann Arbor, MI, USA, 1969. AAI6918131. 32

[87] Marcel F. Neuts. Matrix-geometric solutions in stochastic models - an algorithmic approach. 1982. 32, 33

[88] Guy Latouche and V. Ramaswami. A logarithmic reduction algorithm for quasi-birth-death processes. *Journal of Applied Probability*, 30(3) :650–674, 1993. doi : 10.2307/3214773. 32

[89] Bruce Hajek. Birth-and-death processes on the integers with phases and general boundaries. *Journal of Applied Probability*, 19(3) :488–499, 1982. ISSN 00219002. URL http://www.jstor.org/stable/3213508. 33

[90] Levent Gün and Armand Makowski. Matrix-geometric solution for finite capacity queues with phase-type distributions. pages 269–282, 01 1987. 33

[91] Vittoria de Nitto Persone and Vincenzo Grassi. Solution of finite qbd processes. *Journal of Applied Probability*, 33(4) :1003–1010, 1996. doi : 10.2307/3214981. 34

[92] Essia H. Elhafsi and Mart Molle. On the solution to qbd processes with finite state space. *Stochastic Analysis and Applications*, 25(4) :763–779, 2007. doi : 10.1080/07362990701419946. 34, 82

[93] Jian Xu, Xuefeng Li, Yi Zhong, and Hong Zhang. Availability modeling and analysis of a single-server virtualized system with rejuvenation. *Journal of Software*, 9, 01 2014. doi : 10.4304/jsw.9.1.129-139. 37, 38

[94] Arash Rezaei and Mohsen Sharifi. Rejuvenating high available virtualized systems. In *2010 International Conference on Availability, Reliability and Security*, pages 289–294, 2010. 37, 38

[95] Tuan Anh Nguyen, Dan Kim, and Jong Park. A comprehensive availability modeling and analysis of a virtualized servers system using stochastic reward nets. *The Scientific World Journal*, 2014 :18, 08 2014. doi : 10.1155/2014/165316. 37, 38

[96] Lei HAN and Jian XU. Availability models for virtualized systems with rejuvenation. *Journal of Computational Information Systems*, 9 :8389–8396, 10 2013. doi : 10.12733/jcis8586. 37

[97] Fumio Machida, Dong Seong Kim, and Kishor S. Trivedi. Modeling and analysis of software rejuvenation in a server virtualized system. In *2010 IEEE Second International Workshop on Software Aging and Rejuvenation*, pages 1–6, 2010. 37, 41, 44

[98] Fumio Machida, Dan Kim, and Kishor S. Trivedi. Modeling and analysis of software rejuvenation in a server virtualized system with live vm migration. volume 70, pages 1 – 6, 12 2010. doi : 10.1109/WOSAR.2010.5722098. 37, 39

[99] Hiroyuki Okamura, K. Yamamoto, and TadashiDohi. Transient analysis of software rejuvenation policies in virtualized system : Phase-type expansion approach. *Quality Technology and Quantitative Management*, 11 :335–352, 09 2014. doi : 10.1080/16843703.2014.11673349. 37

[100] May Tar Hla Myint and Thandar Thein. Availability improvement in virtualized multiple servers with software rejuvenation and virtualization. In *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, pages 156–162, 2010. 37

[101] Thandar Thein, Sung-Do Chi, and Jong Park. Availability modeling and analysis on virtualized clustering with rejuvenation. *International Journal of Computer Science and Network Security*, 8(9) :72–80, 01 2008. 37

[102] Thandar Thein and Jong Sou Park. Availability analysis of application servers using software rejuvenation and virtualization. *Journal of Computer Science and Technology*, 24(2) : 339–346, 2009. 37, 38

[103] Matheus Melo, Paulo Maciel, Jean Araujo, Rubens Matos, and Carlos Araújo. Availability study on cloud computing environments : Live migration as a rejuvenation mechanism. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–6, 2013. 37, 38

[104] Jean Rahme and Haiping Xu. A software reliability model for cloud-based software rejuvenation using dynamic fault trees. *International Journal of Software Engineering and Knowledge Engineering*, 25 :1491–1513, 11 2015. doi : 10.1142/S021819401540029X. 37

[105] Fumio Machida, Victor Nicola, and Kishor Trivedi. Job completion time on a virtualized server subject to software aging and rejuvenation. In *2011 IEEE Third International Workshop on Software Aging and Rejuvenation*, pages 44–49, 2011. 37, 38

[106] Fumio Machida, Victor Nicola, and Kishor Trivedi. Job completion time on a virtualized server with software rejuvenation. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10, 01 2014. doi : 10.1145/2539121. 37, 38

[107] Huaming Wu and Katinka Wolter. Software aging in mobile devices : Partial computation offloading as a solution. In *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 125–131, 2015. 38

[108] Yunni Xia, Yakai Han, Mengchu Zhou, and Jia Li. A stochastic model for performance and energy consumption analysis of rejuvenation and migration-enabled cloud. In *Proceedings of the 2014 International Conference on Advanced Mechatronic Systems*, pages 139–144, 2014. 38

[109] Yaxiao Liu, Weidong Liu, Jiaxing Song, and Huan He. An empirical study on implementing highly reliable stream computing systems with private cloud. *Ad Hoc Networks*, 35, 07 2015. doi : 10.1016/j.adhoc.2015.07.009. 38

[110] Antonio Bovenzi, Domenico Cotroneo, Roberto Pietrantuono, and Stefano Russo. Workload characterization for software aging analysis. pages 240–249, 11 2011. doi : 10.1109/ISSRE.2011.18. 39

[111] Antonio Bovenzi, Domenico Cotroneo, Roberto Pietrantuono, and Stefano Russo. On the aging effects due to concurrency bugs : a case study on mysql. 11 2012. doi : 10.1109/ISSRE.2012.50. 39

[112] Selvi Kadirvel and Jose A.B. Fortes. Self-caring it systems : A proof-of-concept implementation in virtualized environments. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 433–440, 2010. 39, 41

[113] Jing Zhao, Yan-Bin Wang, Gao-Rong Ning, Cheng-Hong Wang, and Kishor S. Trivedi andK Y Cai ;Zhen-Yu Zhang. Software maintenance optimization based on stackelberg game methods. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pages 426–430, 2014. 39

[114] Kenichi Kourai and Hiroki Ooba. Zero-copy migration for lightweight software rejuvenation of virtualized systems. In *Proceedings of the 6th Asia-Pacific Workshop on Systems (APSys)*, pages 1–8, 07 2015. doi : 10.1145/2797022.2797026. 39

[115] Matheus Torquato, Paulo Maciel, Jean Araujo, and I. M. Umesh. An approach to investigate aging symptoms and rejuvenation effectiveness on software systems. In *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, 2017. 39

[116] Matheus Torquato, I.M Umesh, and Paulo Maciel. Models for availability and power consumption evaluation of a private cloud with vmm rejuvenation enabled by vm live migration. *The Journal of Supercomputing*, 74 :1–25, 07 2018. doi : 10.1007/s11227-018-2485-4. 39

[117] Mohamed Escheikh, Kamel Barkaoui, and Hana Jouini. Versatile workload-aware power management performability analysis of server virtualized systems. *Journal of Systems and Software*, 125 :365–379, March 2017. doi : 10.1016/j.jss.2016.12.037. URL https://hal.archives-ouvertes.fr/hal-02476550. 40, 52

[118] Armin Zimmermann. Modeling and evaluation of stochastic petri nets with timenet 4.1. pages 54–63, 01 2012. ISBN 978-1-4673-4887-4. doi : 10.4108/valuetools.2012.250263. 44

[119] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow : Enabling innovation in campus networks. *Computer Communication Review*, 38 :69–74, 04 2008. doi : 10.1145/1355734.1355746. 69

[120] Giuseppe Lettieri, Vincenzo Maffione, and Luigi Rizzo. A survey of fast packet i/o technologies for network function virtualization. pages 579–590, 10 2017. ISBN 978-3-319-67629-6. doi : 10.1007/978-3-319-67630-2_40. 71

[121] Data Plane Development Kit (DPDK), 2018. URL https://www.dpdk.org/about/. 71

[122] Christos Langaris. Gated polling models with customers in orbit. *Mathematical and Computer Modelling*, 30 :171–187, 08 1999. doi : 10.1016/S0895-7177(99)00140-5. 71

[123] Hideaki Takagi. Queuing analysis of polling models. *ACM Comput. Surv.*, 20(1) :5–28, March 1988. ISSN 0360-0300. doi : 10.1145/62058.62059. URL https://doi.org/10.1145/62058.62059. 71, 72

[124] Hideaki Takagi. 71

[125] Hanoch Levy and Moshe Sidi. Polling systems : applications, modeling, and optimization. *IEEE Transactions on Communications*, 38(10) :1750–1760, Oct 1990. ISSN 1558-0857. doi : 10.1109/26.61446. 71

[126] Tony T. Lee. M/g/1/n queue with vacation time and exhaustive service discipline. *Oper. Res.*, 32(4) :774–784, August 1984. ISSN 0030-364X. 72

[127] Tony T. Lee. M/g/1/n queue with vacation time and limited service discipline. *Performance Evaluation*, 9(3) :181–190, August 1989. URL https://doi.org/10.1016/0166-5316(89)90025-4. 72

[128] Phuoc Tran-Gia and Thomas Raith. Performance analysis of finite capacity polling systems with nonexhaustive service. *Perform. Eval.*, 9(1) :1–16, November 1988. ISSN 0166-5316. doi : 10.1016/0166-5316(88)90021-1. URL http://dx.doi.org/10.1016/0166-5316(88)90021-1. 72, 73

[129] Phuoc Tran-Gia and Thomas Raith. Performance analysis of finite capacity polling systems with nonexhaustive service. *Perform. Eval.*, 9(1) :1–16, November 1988. ISSN 0166-5316. doi : 10.1016/0166-5316(88)90021-1. URL http://dx.doi.org/10.1016/0166-5316(88)90021-1. 73

[130] Daniel Kofman. Blocking probability, throughput and waiting time in finite capacity polling systems. *Queueing Systems*, 14(3) :385–411, Sep 1993. ISSN 1572-9443. doi : 10.1007/BF01158875. URL https://doi.org/10.1007/BF01158875. 73

[131] Adnan Sohail. Performance evaluation of a non-exhaustive polling system with asymmetrical finite queues. In *2012 UKSim 14th International Conference on Computer Modelling and Simulation*, pages 613–617, March 2012. doi : 10.1109/UKSim.2012.94. 73

[132] Guillaume Artero Gallardo, Bruno Baynat, and Thomas Begin. Performance modeling of virtual switching systems. In *IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2016*, pages 125–134, London, United Kingdom, September 2016. IEEE. doi : 10.1109/MASCOTS.2016.22. URL https://hal.archives-ouvertes.fr/hal-01387726. 4, 73, 75, 83, 90, 92

[133] Levy Yonatan and Yechiali Uri. Utilization of idle time in an m/g/1 queueing. *system.Manage. Sci*, 22(2) :202–211, Oct 1975. doi : 10.1287/mnsc.22.2.202. 74

[134] Doshi Bharat T. Queueing systems with vacations - a survey. *Queueing Systems*, 1(1) :29–66, Jun 1986. ISSN 1572-9443. doi : 10.1007/BF01149327. URL https://doi.org/10.1007/BF01149327. 74

[135] Hideaki Takagi. Queueing analysis a foundation of performance evaluation volume 1 : Vacation and priority systems. 1, 01 1991. 74

[136] Naishuo Tian and Zhe George Zhang. *Analysis and Application of Polling Models*. Springer, Boston, MA, 2006. ISBN 978-0-387-33721-0. 74

[137] Les Servi and Steven Finn. M/m/1 queues with working vacations (m/m/1/wv). *Performance Evaluation*, 50 :41–52, 10 2002. doi : 10.1016/S0166-5316(02)00057-3. 74

[138] Medhi Jyotiprasad. *Stochastic Processes*. Whily, Eastern, 1989. 74

[139] Neuts Marcel F. A general class of bulk queues with poisson input. 1967. 74

[140] Moeko Yajima and Tuan Phung-Duc. Batch arrival single server queue with variable service speed and setup time. *Queueing Systems*, 05 2017. doi : 10.1007/s11134-017-9533-2. 74

[141] Manfield. D and Phuoc Tran-Gia. Analysis of a finite storage system with batch input arising out of message packetization. *IEEE Transactions on Communications*, 30(3) :456–463, March 1982. ISSN 0090-6778. doi : 10.1109/TCOM.1982.1095495. 74

[142] Branislav Dragovic, Nam Kyu Park, Nenad Zrnic, and Romeo Mestrovic. Mathematical models of multiserver queuing system for dynamic performance evaluation in port. *Mathematical Problems in Engineering*, 2012, 01 2012. doi : 10.1155/2012/710834. 75

[143] Yutaka Baba. On the mx/g/1 queue with vacation time. *Operations Research Letters*, 05 : 93–98, July 1986. doi : 10.1016/0167-6377(86)90110-0. 75

[144] Gautam Choudhury. Analysis of the mx̂/g/1 queueing system with vacation times. *Series B*, 64 :37–49, 01 2002. doi : 10.2307/25053201. 75

[145] Yutaka Baba. The m^x/m/1 queue with multiple working vacation. *American Journal of Operations Research*, 02 :217–224, 01 2012. doi : 10.4236/ajor.2012.22025. 75

[146] Karabi Sikdar and U. Gupta. On the batch arrival batch service queue with finite buffer under server's vacation : Mx/gy/1/n queue. *Computers & Mathematics with Applications*, 56 :2861–2873, 12 2008. doi : 10.1016/j.camwa.2008.07.034. 75

[147] D. Manfield and Phuoc Tran-Gia. Analysis of a finite storage system with batch input arising out of message packetization. *IEEE Transactions on Communications*, 30(3) :456–463, March 1982. ISSN 0090-6778. doi : 10.1109/TCOM.1982.1095495. 86, 87

[148] ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/169862. 87

Glossaire

ACPI	Advanced Configuration and Power Interface États de gestion de l'énergie
API	Application Programming Interface Interface de Programmation d'Application
CCS	Calculus of communicating systems Calcul des Systèmes de Communication
CAPEX	CAPital EXpenditure dépenses d'investissement
CPU	Central Processing Unit Cœurs de Processeur
CTMC	Continuous-Time Markov Chains Processus de Markov à temps continu
DFT	Dynamic Fault Trees les arbres de défaillances dynamiques
DPDK	Data Plane Development Kit
DPM	Dynamic Power Management Gestion Dynamique de l'Alimentation
DSI	Directions des Systèmes d'Information
DVFS	Dynamic Voltage Frequency scaling Passage à l'échelle dynamique de la tension et de la fréquence
EC2	Elastic Compute Cloud
ETSI	Institut européen des normes de télécommunications
FIFO	First In First Out Premier Arrivé, Premier Sorti
GCE	Google Compute Engine
GQBD	Generalised Quasi Birth and Death

DBD	Quasi Naissance et de Mort généralisé
GSPN	Generalized Stochastic Petri Nets Réseaux de Petri Stochastiques Généralisés
IaaS	Infrastructure-as-a-Service
IDC	Indice de dispersion Index of dispersion for counts
LIFO	Last In First Out Dernier Arrivé, Premier Sorti
MBS	Mean Batch Size Taille moyenne du lot
MMPP	Markov Modulated Poisson Process Processus de Poisson modulé de Markov
M&O	Management and Orchestration Gestion et Orchestration
MRSPN	Markov Regenerative Stochastic Petri Nets Réseaux de Petri Stochastiques Régénératifs de Markov
NFV	Network Functions Virtualization Virtualisation des Fonctions Réseau
NFVI	Network Function Virtualisation Infrastructure Infrastructure de virtualisation de fonction de réseau
NIC	Network Interface Cards Cartes d'interface de réseau
NIST	National Institute of Standards and Technology Institut national des normes et de la Technologie
OIE	Observation, impacts et énergie Observation, impacts et énergie
OF	OpenFlow

ONF	Open Networking Foundation Fondation des Réseaux Ouverts
OPEX	OPerational EXpenditure Dépenses d'exploitation
OS	Operating System Système d'Exploitation
OvS	Open vSwitch
PaaS	Plateform-as-a-Service
PMC	Power Manageable Component Composant de Gestion de l'Alimentation
PS	Processor Sharing Partage du Processeur
QBD	Quasi Birth and Death Quasi Naissance et de Mort
RBD	Reliability Block Diagrams Diagrammes de Fiabilité
RdP	Réseau de Petri
RdPC	Réseau de Petri Colorés
SaaS	Software-as-a-Service
SAR	Software Aging Rejuvenation Rajeunissement du vieillissement des logiciels
SAN	Stochastic Automata Networks Réseaux d'automates stochastiques
SDN	Software-Defined Networking(SDN Réseaux Définis par Logiciels
SLA	Service Level Agreement Accord de niveau de service

SMP	Semi- Markov Processes Processus Semi-Markov
SPN	Stochastic Petri Net Réseaux de Petri Stochastiques
SRN	Stochastic Reward Nets Réseaux de Récompenses Stochastiques
SVSs	Server Virtualized Systems Systèmes de Serveurs Virtualisés
TCA	Total Cost of Acquisition coût total d'acquisition
TCO	Total Cost of Ownership coût total de possession
VM	Virtual Machine Machine Virtuelle
VMM	Virtual Machine Monitor Moniteur de Machine Virtuelle
VMSR	Virtual Machine based Software Rejuvenation Machine Virtuelle basés sur Rajeunissement des Logiciels
VNF	Virtualized Network Function Fonction de Réseau Virtualisée