CONTEXTE

Un datacenter est un site physique sur lequel se trouvent regroupés des équipements constituants du système d'information de l'entreprise : ordinateurs centraux, serveurs, etc.

L'architecture d'un dc est représentée ainsi. Mais ces dernières années ont vu la virtualisation s'imposer comme technologie de base dans les DCs (), car elle permet de tasser le maximum de services sur un nombre réduit de machines physiques. Ainsi les machines physiques non utilisées sont éteintes ce qui réduit la consommation électrique qui représente 50 à 70% des dépenses dans un DC

---

les entreprises et les particuliers achètent auprès des fournisseurs de datacenters des espaces pour faire tourner leurs applications. Le pb est qu'ils ont tendance à reserver bien plus de ressources que ce dont ils ont réellement besoin ceci par souci de performance, ce qui a pour conséquences…. Ce qui entraine une perte d'argent

Le client achète principalement : l'espace disque et l'espace mémoire et c'est la mémoire qui est la ressource critique dans un DC
Ainsi pour résoudre les pbs présentés et surtout faire plus de gains, les gestionnaires de DCs cherchent à savoir pour chaque application hébergée quelle est la quantité de mémoire dont elle a effectivement besoin au cours de son exécution : cette quantité s'appelle le working set size (ceci permettra d'allouer de la mémoire dynamiquement, i.e. au prorata des besoins)

PROBLEMATIQUE

Le pb d'estimation du wss ne date pas d'aujourd'hui. Il existe déjà de nombreux travaux qui s'y sont intéressés et des techniques d'estimation qui en découlent. Ces techniques sont basées sur des approches et méthodologies qui … ces points sont des inconvénients car ils peuvent avoir des impacts négatifs sur les applications des clients : lenteur, indisponibilité, etc. ceci est inacceptable pour le client car l'estimation du wss profite au provider, il doit dc pouvoir le faire sans perturber l'exécution des applications des clients
Le pb tourne donc autour de la question suivante :

MOTIVATIONS ET OBJECTIFS

Depuis 2016 Intel en collaboration avec VMWare (développeur d'hyperviseur, nous verrons par la suite ce qu'est un hyperviseur) a commencé à produire des processeurs équipés d'une nouvelle technologie de virtualisation appelée Page Modification Logging, en abrégé PML
Le pml a été introduit par Intel pour faciliter l'obtention des statistiques liées au working set (telles que les zones de la mémoire que la machine utilise pendant son exécution, nous y reviendrons plus en détails dans la suite)

Nous avons porté un intérêt particulier à cette fonctionnalité pour ++ raisons :

- Une technique d'estimation du wss basée sur le mécanisme pml aurait ceci de différent que c le processeur lui mm qui nous renvoie les informations nécessaires à l'estimation : on parlera de solution matérielle (le matériel faisant ici référence au processeur)
- Ceci nous permettra donc de pallier les pbs que posent les techniques existantes qui elles sont basées sur des approches dites logicielles (nous expliquerons pourquoi quand nous présenterons ces techniques par la suite)

- En outre, jusqu'ici aucun travail de recherche ne s'est intéressé à cette nouvelle fonctionnalité

Nous avons entrepris ce travail avc pour but :

- Étudier le design actuel du pml afin de mettre en exergue les limites que présente le mécanisme pour l'estimation du ws
- Proposer une nouvelle architecture mieux adaptée pour répondre au pb posée dans la problématique
- Définir un algorithme d'estimation qui s'appuie sur cette nouvelle architecture
- Evaluer notre technique et la comparer à celles qui existent déjà

GENERALITES SUR LA VIRTUALISATION

La figure svte représente l'architecture d'un syst virtualisé où on a les machines virtuelles au dessus du matériel de la machine hôte

prenons l'exemple d'une femme enceinte : quand la maman se déplace, le bb se déplace aussi mais pas en marchant coe le fait la maman, le bb respire et se nourrit mais à travers les fonctions vitales et digestives de la maman si la maman arrête de respirer, le bb va suffoquer, si la maman ne sourit plus le bb en souffre ; et bien ds cet exemple le bb est comparable à la machine virtuelle et la maman à la machine physique ou hôte; une machine virtuelle c juste son syst d'exploitation et ses applications, elle utilise le matériel (processeurs, mémoires, claviers, etc.) de la machine physique

Comme on peut le voir sur la fig entre les vm et le matériel nous avons l'hyperviseur qui est l'outil (logiciel) qui nous permet de faire tourner toutes ces machines virtuelles dans notre machine physique. En revenant à l'exemple de la femme enceinte, nous pouvons assimiler l'hyperviseur au cordon ombilical qui lie le bb à sa maman. Tout comme le cordon permet la circulation des nutriments par exemple de la mère à l'enfant, l'hyperviseur permet d'allouer les ressources physiques aux machines virtuelles

Dans ce travail nous utilisons le syst de virtualisation XEN qui est utilisé par Amazon dans ses DCs, il est assez connu et il a déjà modifié son code pour prendre en charge le mécanisme du PML. l'architecture d'un syst virtualisé avc xen est le suivant; ici la machine hôte est elle mm considérée comme une vm mais avc ts les privilèges : dom0, et les autres vm st des domU pour domain unprivileged

VIRTUALISATION ASSISTEE PAR LE MATERIEL

Aujourd'hui le concept de virtualisation s'étend à la mémoire et au processeur : on parle de hardware assisted virtualization ou virtualisation assistée par le matériel
Intel a introduit dans ses processeurs des fonctionnalités pour prendre en compte ce niveau de virtualisation : VT-x pour la virtualisation du processeur et EPT pour la virtualisation de la mémoire

VIRTUALISATION DE LA MEMOIRE : ENVIRONNEMENT NATIF

Dans un environnement natif, i.e. sans système de virtualisation, la mémoire centrale est divisée en espaces appelés pages mémoire. Lorsque le processeur exécute un processus, celui-ci utilise

des pages avec des adresses dites virtuelles. Ce n'est q lorsq le processus essaiera de modifier une de ces pages que le processeur fera appel à l'unité de gestion mémoire pour trouver les adresses réelles correspondantes. En effet, la mmu maintient pour chaque processus, une table de pages qui contient les correspondances entre les adresses virtuelles des pages du processus et les adresses physiques en mémoire centrale
C dc ainsi que la mémoire est virtualisée dans un environnement natif

Étant donné que la table de pages est en mémoire centrale, cela ralentirait le processeur d'aller chercher les adresses virtuelles en mémoire centrale à chaque écriture. Ainsi, il existe à l'intérieur du processeur mm une structure de données qui contient les entrées récentes de la table de pages : la TLB. de cette façon, si l'adresse recherchée se trouve dans la tlb, le processeur peut directement aller à l'emplacement mémoire correspondant. Ds le cas où aucune entrée n'est trouvée on parle de **tlb miss**. Et le processeur reprend le processus normal décrit précédemment

VIRTUALISATION DE LA MEMOIRE : ENVIRONNEMENT VIRTUALISATION (EPT)

Dans un environnement virtualisé, chaque vm est un syst d'exploitation à part entière comme ns l'avons mentionné plus haut; donc tout comme dans un environnement natif chaque processus maintient une table de pages qui contient les correspondances entre adresses virtuelles et physiques : gPT

Ici les adresses virtuelles des pages des processus sont appelées gVA. Lorsqu'un processus s'exécutant dans la machine virtuelle veut modifier une page, il faut trouver en mémoire centrale l'adresse réelle y correspondant appelée hPA. Ici la translation se fait à 2niveaux : 1rement ds la vm avc le gPT, on obtient une adresse physique dans la vm appelée gPA, ensuite dans l'hyperviseur avec la table de page étendue ou EPT qui donne la translation guest physical à host physical

VIRTUALISATION DU PROCESSEUR : INTEL VT-X

Pour mieux comprendre le PML et son mécanisme de fonctionnement il est nécessaire de connaître certaines notions liées au support des processeurs Intel pour la virtualisation

Les extensions vt-x dupliquent entièrement l'état architectural visible du processeur afin que la machine virtuelle aie l'illusion d'y avoir complètement accès

Pour que ceci ne reste qu'une illusion, un nouveau mode d'exécution est introduit : le mode root. Ainsi, l'os hôte et l'hyperviseur s'exécutent en mode root, i.e. avc ts les droits et privilèges d'accès au matériel, et les autres vms s'exécutent en mode non-root avec des accès restreints

Avec les vt-x chaque vm a en mémoire centrale une structure de données gérée par l'hyperviseur et appelée vmcs, virtual machine control structure, qui sert à sauvegarder l'état de la VM.
Lorsque l'état de la vm est chargé, elle s'exécute en mode non-root jusqu'à ce qu'elle génère une exception qui doit être prise en main par l'hyperviseur : cette transition du mode non-root au mode root s'appelle #vmexit

PML : PAGE MODIFICATION LOGGING

Il est tout d'abord à noter que le pml n'est utilisé que dans le cadre de la virtualisation; ce mécanisme ne peut dc pas être utilisé dans un environnement natif. Le pml étend...

Nous allons dc expliquer comment : intel a introduit dans ses processeurs récents des bits accessed et dirty pour l'ept, de sorte que quand la vm accède à une page le bit accesed est mis à 1, et lorsque la page est modifiée le bit dirty est mis à 1.

Le pml se base sur ces bits accessed et dirty, en étendant le procédé qui se produit lorsqu'ils sont mis à jour. En effet, lorsqu'une écriture fait passer le bit dirty d'une page de 0 à 1, le processeur va enregistrer l'adresse physique de cette page (gPA) dans un espace de logs appelée page modification log (PMLog). Nous présenterons les détails de ce mécanisme dans la suite

L'introduction du pml a nécessité des changements dans la structure de contrôle vmcs de la machine virtuelle. Ceux qui nous intéressent sont les champs suivants :

LES METRIQUES DE L'ESTIMATION

Comme nous l'avons mentionné plus haut, la mémoire est la ressource critique en terme de consolidation de vms.

Pour pallier ce pb de gestion de la mémoire, la solution la plus communément utilisée est l'allocation de mémoire à la demande, qui consiste à : collecter périodiquement les informations sur l'activité de la vm et s'en servir d'input pour estimer la quantité de mémoire dont elle a besoin, et une fois l'estimation faite ajuster la mémoire allouée à la vm en conséquence

Cette approche nécessite de répondre à 2 interrogations : Q1 & Q2;
De ces 2questions découlent les métriques à prendre en compte lors de la définition d'une technique d'estimation du ws

Répondre à Q1 soulève 2 défis :

- Le 1er relatif à la méthode utilisée pour collecter les données liées à l'activité de la vm. Cette méthode ne doit pas être active, i.e. ne doit pas modifier le cours d'exécution de la vm
- Le 2e se rapporte au niveau d'implémentation de la méthode utilisée. Elle peut se faire soit à exclusivement à l'intérieur de l'hyperviseur (ou du dom0), soit exclusivement à l'intérieur de la VM soit répartie à travers les 2. Ds ces 2derniers k la méthode est dite intrusive et nécessite l'accord du client, sauf pour des datacenters privés

Quant à Q2 les défis sont :

- La précision de l'estimation : étant donnée que la mémoire allouée à la vm est réajustée après l'estimation, une estimation erronée pourrait soit impacter sur les performances de la machine virtuelle (en la surchargeant), soit causer un gaspillage encore plus important de la mémoire
- Les coûts liés à l'algorithme : en effet un algorithme qui demande énormément de ressources au processeur entraînerait une surcharge de l'hyperviseur et du dom0

TECHNIQUES EXISTANTES

Nous allons maintenant présenter quelques techniques existantes en montrant pr chacune d'elles comment elle répond aux questions Q1 et Q2, et ensuite ses caractéristiques et limites

Self-ballooning

Self-ballooning repose essentiellement sur la vm, en particulier son syst d'exploitation.

- Réponse à Q1 : pour répondre à Q1, self-ballooning considère que le wss de la vm est donnée par les statistiques du noyau à l'occurrence la valeur du committed_as qui correspond au nombre total de pages virtuelles allouées par un processus, même si elles ne correspondent pas forcément à des pages physiques en mémoire centrale. Pour obtenir cette valeur il suffit de taper sur un terminal
- Réponse à Q2 : pour réajuster la mémoire de la vm, l'OS décrémente la valeur du committed_as lorsque des pages allouées à la vm sont libérées, et l'incrémente lorsqu'un programme fait une allocation de mémoire
- Caractéristiques et Limites :
    - Cette méthode est intrusive car c le balloon driver situé à l'intérieur de la vm qui se charge d'augmenter ou de diminuer la mémoire allouée à la vm. Le balloon driver est un dispositif logiciel dont l'hyperviseur équipe chaque machine virtuel et qui peut être gonflé ou dégonglé dans le but d'augmenter ou de diminuer la mémoire de la machine
    - En outre la valeur du committed_as ne prend en compte que la mémoire cache, celles des pages virtuelles allouées à un processus, dc une estimation basée sur cette valeur sera la plupart du temps supérieure à la mémoire activement utilisée par la machine, ce qui conduira à un gaspillage encore plus conséquent de ressources

Vmware

Vmware s'appuie sur une approche d'échantillonnage.

- Réponse à Q1 : pour répondre à Q1, l'hyperviseur choisit aléatoirement et périodiquement n pages de la mémoire de la vm et les rend invalides soit en les marquant comme étant non présentes (i.e. en mettant leur bit de présence dans la table de pages à 0), soit en les marquant comme read_only i.e. en lecture seule, ainsi lorsqu'un programme essaiera d'accéder ou de modifier ces pages, cela génèrera une exception qui sera capturée par l'hyperviseur
- Réponse à Q2 : une fois l'exception générée, l'hyperviseur compte le nombre f de pages de l'échantillon qui ont été sujettes à des défauts de page (i.e. des pages marquées non-présentes ou read_only auxquelles la vm a essayé d'accéder ou d'effectuer une écriture).

  De cette façon, vmware estime le wss par la formule $(f/n)*m_{act}$, où m_{ac} est la mémoire actuellement allouée à la vm
- Caractéristiques et Limites :
    - Non intrusive car entièrement implémentée dans l'hyperviseur
    - Toutefois elle est active car le fait d'invalider les pages modifie le cours d'exécution de la vm en provoquant des défauts de pages

- Le coût de résolution de ces défauts de page peut entraîner en outre des dégradations de performances
- Notons également que vmware est incapable d'estimer un ws supérieure à la mémoire actuellement allouée à la vm car l'échantillon est pris à partir de celle-ci. Dc au pire des cas, la technique déterminera q toutes les pages ont été accédées et estimera dc le ws à la valeur actuelle de la mémoire allouée

Geiger

- Réponse à Q1 : pour observer l'activité mémoire de la vm, geiger observe les évictions et mises à jour éventuelles des pages du cache de la vm
- Réponse à Q2 : geiger repose sur une technique appelée buffer fantôme. C'est un buffer imaginaire qui étend la mémoire physique actuellement allouée à la vm. Sa taille représente la qté de mémoire supplémentaire à allouer à la vm pour éviter qu'elle n'effectue des swaps de pages. L'estimation se fait dc à travers la formule :
- Caractéristiques et Limites :
  - Tout comme vmware, geiger est totalement transparente du point de vue de la vm
  - Toutefois ce caractère pose un sérieux pb car la technique n'est capable d'estimer le ws que si la taille du buffer fantôme est strictement positive, i.e. si la vm est ds un état de swap. Dc contrairement à wmware geiger devient inefficace si le ws de la machine est inférieure à la mémoire qui lui est actuellement allouée

ARCHITECTURE ACTUELLE DU PML

Comme nous l'avons dit à l'introduction, aucun travail de recherche n'a jusqu'ici été mené sur le pml, donc notre 1re contribution a été de mettre sur pied l'environnement nécessaire pour pouvoir l'activer et voir comment il fonctionne. C'est pourquoi nous allons tt d'abord expliquer ce fonctionnement.

Avant de le faire, prenons un exemple pour resituer le problème : considérons le département du gi dans lequl nous nous trouvons actuellement et qu'un jr de cours le chef de dépt veuille savoir dans chaque salle cours combien d'étudiants il y a, si le prof est là et de quoi il a besoin… une solution naïve serait d'arpenter les couloirs, de passer salle par salle, frapper et entrer pour obtenir ces informations. Seulement si on suppose que lorsqu'un enseignant est en salle il en le maître et qu'il la ferme à clés, le fait q le chf de dpt frappe à la porte perturbe le cours, freine l'enseignant et peut faire perdre le fil aux étudiants. La salle de classe fermée à clés ici représente la vm qui est une boite noire ie qu'on ne peut rien en tirer de l'extérieur, le fait de perturber le cours représente les limites des solutions actuelle, l'intrusion la modification du cours d'exécution de la vm… au lieu de cela, le chf de dpt pourrait décider de mettre dans chaq salle un surveillant qui lui enverrait toutes les heures par exple les informations dt il a besoin, mm les noms des étudiants qui dorment pendant le cours… bref le fait d'installer un surveillant ici est assimilable au mécanisme du pml, pour nous éviter de nous introduire dans la machine, le pml va nous renvoyer les informations dont nous avons besoin

Considérons les vms domU1 et domU2 qui s'éxécutent sur ces processeurs, si le PML est acitvé pour cette machine virtuelle alors quand elle modifie …
Il ns ns restera plus qu'à aller en mémoire et lire cette bitmap pour savoir quelles pages la vm utilise

LIMITES DE L'ARCHITECTURE ACTUELLE

Notre 2ème contribution a été de ressortir les limites que présente le mécanisme actuel pour l'estimation du ws.

Limite 1: vmexit imputé à la vm lorsq le pmlog est plein
Comme nous l'avons expliqué lorsque la page de log est pleine, le processeur génère un vmexit qui stoppe l'exécution de la vm. Il y a changement de contexte (passage du mode non-root au mode root) et la main passe à l'hyperviseur. Or ces transitions imposent des overhead qui sont inacceptables pour le client. N'oublions pas q les coûts liés à l'estimation du ws ne doivent pas impacter l'exécution des applications des clients, or pdt le vmexit la machine arrête de s'exécuter, mm si pdt une nanoseconde en terme de tmps processeur cela pour être crucial en terme de performances

Limite 2 : la taille du pml_log qui est de 4KB
Avec cette taille, le pml log ne peut contenir que 512 entrées de 64bits, ce qui est très petits au regard de la plupart des applications de nos jours. La conséquence de cette petite taille est qu'elle accroît le nombre d'évènements pml_log full et donc le nombre de vmexits imputés à la vm

Limite 3 : le pml logue uniquement les adresses des pages modifiées
Le pml n'enregistre une adresse que si son bit dirty passe de 0 à 1, il s'agit donc uniquement des pages qui sont modifiées. Or le working set inclut toutes les pages utilisées par la machine que ce soit opération d'écriture ou de lecture. Donc pour l'instant le pml passe à côté des pages auxquelles la vm accède sans les modifier et donc néglige une bonne partie du working set surtout pour les charges de travail de lecture

Limite 4 : le pml ne tient pas compte de la chaleur des pages
La chaleur d'une page peut être vue simplement comme le nombre de fois où elle est utilisée. Une page est dite chaude si elle est constamment utilisée par le système. Comme nous l'avons dit, une adresse est loguée si son bit dirty passe de 0 à 1. Ceci implique que si une adresse a déjà été enregistrée, si elle modifiée ultérieurement elle ne sera plus prise en compte en compte étant donné que son bit dirty est déjà à 1 et que le matériel ne le remet pas à 0. Et quand bien mm le bit serait remis à 0 de façon logicielle et serait enregistrée de nouveau, dans la structure bitmap lorsque le bit correspondant à une adresse est déjà à 1 rien n'est fait. Le pb est q toutes les pages utilisées par le syst ne ft pas forcément partie de son ws, il faudrait dc connaître le nombre de fois que ces enregistrées sont utilisées

Limite 5 : le pml enregistre également les adresses des pages de la table de pages

En cas de tlb miss (nous l'avons expliqué précédemment, lorsqu'une adresse ne se trouve pas ds la tlb), le processeur doit aller chercher en mémoire centrale l'adresse physique correspondant au guest physical address à l'origine du tlb miss. Pour cela il va devoir effectuer un parcours de l'ept comme sur cette figure. Lorsqu'une page physique est modifiée le pml enregistre le gpa à l'origine de cette modification, or pdt cette translation d'adresse il y a 4pages physiques intermédiaires qui seront modifiées donc 4gpa supplémentaires qui seront loguées, ces adresses sont dites superflues et remplissent inutilement le pml log et donc concourrent à augmenter le nombre vmexits.

PROPOSITION D'UNE ARCHITECTURE AMELIORÉE

En fonction des limites évoquées plus haut, nous avons pensé un nouveau design architectural dans le but d'améliorer le mécanisme actuel du PML afin qu'il réponde mieux aux besoins d'estimation du WSS.

1. Redirection des vmexits

Cette amélioration permettra de pallier la limite 1. Ainsi, lorsque que le pml_log est plein, nous proposons que le processeur envoie un signal non plus à l'hyperviseur mais au dom0 qui se chargera de traiter cette interruption

2. Cette deuxième amélioration répond à la limite 2, à savoir la taille insuffisante du page modification log.

En outre, elle vient en appoint à la première à savoir le fait que l'on redirige les VMExits vers le dom0. En effet, lorsque le pml_log sera plein, la VM ne va plus arrêter son exécution, or comme nous l'avons mentionné dans la description du mécanisme actuel, lorsque le pml_log est plein aucune adresse supplémentaire ne peut être enregistrée dans ce dernier, c pourquoi la machine arrêtait son exécution. Maintenant q ce n'est plus le cas, pour éviter de perdre des adresses, nous proposons de rajouter un deuxième page modification log, qui pourra continuer d'enregistrer des logs pendant que le premier est vidé. De cette façon, lorsque ce deuxième buffer sera plein, il passera de nouveau la main au premier et ainsi de suite

3. Comme nous l'avons expliqué, une fois que le bit dirty d'une page passe de 0 à 1, le matériel ne le remet pas à 0. Une solution naïve serait de parcourir la table de pages et remettre le bit dirty à 0, mais ceci est une opération très coûteuse. Nous proposons donc une amélioration qui consisterait à enregistrer également par le matériel l'entrée de la table de pages correspondant à l'adresse enregistrée. En ayant cette information, il sera aisé de remettre à 0 les bits des pages en évitant celles de la table de pages (ce qui répond donc à la limite).

4. Lorsque le pml_log est plein, avant de réinitialiser son index, les informations qu'il contient doivent être consignées dans la bitmap comme expliqué précédemment. Avec la structure actuelle qu'elle a, i.e. constituée uniquement de bits, il nous est difficile d'ajouter les informations dont nous avons besoin.
Nous proposons donc de modifier cette structure de consolidation des logs de sorte qu'elle contienne :

- Les adresses des pages modifiées et leur nombre d'occurrences
- Les adresses des pages lues et leur nombre d'occurrences

Elle ne sera donc plus constituée de bits mais de nombres avec le type long, d'où la nvelle appellation longmap. Ceci permettra de répondre aux limites 3 et 4

AVANTAGES ET INCONVENIENTS

Nous avons vu dans la partie précédente que toute technique d'estimation devait répondre à 2 questions :

La 1re : comm….
Dans notre technique c'est le matériel lui mm qui se charge de collecter les informations sur l'activité de la vm, en traquant les pages de la vm et en consignant les adresses de celle-ci dans une structure de consolidation de logs. Ce qui rend notre méthode totalement transparente du point de vue de la vm et donc non intrusive

Actuellement, le coût facturé à la VM concerne l'ensemble des VMExits qui lui sont imposés lorsque le pml_log est plein. La solution architecturale que nous proposons vient palier cette limite en redirigeant ces interruptions vers le dom0, donc aucune surcharge ne sera imposée ni à la VM ni à l'hyperviseur

Quant à la 2ème question
Il nous suffit de compter le nombre n de pages et connaissant la taille d'une page (4KB), il n'y a plus qu'à effectuer l'estimation à travers la formule
Etant donné que c'est le matériel lui-même qui se charge de récupérer les adresses des pages utilisées par la VM, on peut être sûr qu'aucune adresse ne sera manquée, et donc de la précision de l'estimation. Pour l'instant, cette précision de calcul ne peut être vérifiée car il faudrait d'abord que la nouvelle architecture soit mise sur pied.

DETAILS D'IMPLÉMENTATION

Nous avons implémenté, parmi les propositions d'améliorations faites, celles qui ne nécessite pas de modification matérielle

- Hypercalls d'activation et désactivation du pml

Même s'il est supporté par le processeur, le mécanisme du pml n'est pas activé par défaut.
L'hyperviseur xen que nous utilisons, a déjà intégré le mécanisme ds sn code source mais uniquement dans le cadre de la migration des machines virtuelles pour déterminer les pages chaudes qui doivent être déplacées pendant la migration. Or dans le cadre de notre travail, nous devons être capables d'activer le PML hors de ce contexte, i.e. chaque fois que nous avons besoin d'estimer le WSS d'une VM. nous avons donc défini des commandes xl : xl enable_log_dirty et xl disable_log_dirty

- Modification de la structure de données bitmap

Actuellement la structure de consolidation des logs est un radix tree dont chaque branche est constituée de 3noeuds L4, L3 et L2 et d'une feuille terminale L1. les entrées de chaque noeud sont des pointeurs vers le niveaux suivant et une entrée dans une feuille est 1bit représentant une adresse. Sur la base de calculs effectués, nous avons rajouté un niveau L0 à l'arbre de sorte q les entrées du niveau L1 pointent dorénavant vers une page L0 qui contiendra les logs sous le format adresse-nb_d_occurrences. Nous l'appelons maintenant longmap

- Modification du traitant pml_buffer_full

Nous avons modifié le comportement du mécanisme lorsque le pml_log est plein en accord avec la modification de la structure de consolidation de logs précédente. Ainsi, au moment de vider le buffer, l'algorithme parcours la longmap, et s'il existe déjà une entrée pour l'adresse à enregistrer, son compteur est incrémenté. Sinon, une nouvelle entrée est créée avec un compteur initialisé à 1 pour cette.

- Mécanisme de copie des logs consolidés de l'hyperviseur vers le dom0

Pour éviter d'imposer des coûts de surcharge à l'hyperviseur, nous implémentons les algos de calcul dans le dom0. Pour cela, il faut copier les logs de la vm depuis la longmap dans l'hyperviseur, vers le dom0. Ceci ne peut se faire qu'à l'aide d'un hypercall que nous avons donc défini : «xl collect¡ dirty¡ logs».

EXPERIMENTATIONS

Les expérimentations ont été menées avec des charges synthétiques dont la variation de mémoire en fonction du temps est présentée sur ces graphes
Ces charges manipulent 400MB de mémoire soit 400*1024/4=102400 pages mémoire. Soit de façon cte, soit de façon variable.

EXPERIMENTATION 1

Nous avons fait tourner dans la vm une application manipulant activement 102400 pages mémoire de façon constante et pendant son exécution nous avons estimé le ws de la machine au moyen de 5 techniques d'estimation existantes puis avec celle que nous avons implémentée.

Les résultats sont présentés sur ces diagrammes. Seules les techniques geiger et exclusive-cache donnent une estimation exacte. Quant à la solution basée sur le pml, elle permet de détecter 105000 pages soit un ws de 410MB. Ce surplus de 10MB s'explique entre autres par la limite 5 que nous avons évoquée prédemment à savoir que le mécanisme actuel du pml enregistre également les pages de la table des pages,

EXPERIMENTATION 2

Le but de cette expérimentation est de vérifier que les techniques utilisées sont capables de détecter les variations dans l'utilisation de la mémoire. En effet, rares sont les applications qui auront des ws constants. Les ressources consommées par la plupart des applications varient au cours de leur exécution

Les résultats sont présentés sur les graphes ci-après

Comme nous les montrent ces résultats, il est difficile pour la plupart des techniques de donner une bonne estimation lorsque le ws de la machine devient variable. Ce qui n’est pas le cas avec la solution implémentée