Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Μηχανισμός και πολιτικές για επιλογή και δυναμική εναλλαγή μεθόδων σελιδοποίησης σε εικονικοποιημένα περιβάλλοντα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΣΩΝ ΜΑΡΜΑΝΗΣ

Επιβλέπων : Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2020

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Μηχανισμός και πολιτικές για επιλογή και δυναμική εναλλαγή μεθόδων σελιδοποίησης σε εικονικοποιημένα περιβάλλοντα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΣΩΝ ΜΑΡΜΑΝΗΣ

Επιβλέπων : Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 2η Ιουλίου 2020.

Γεώργιος Γκούμας Επ. Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Διονύσιος Πνευματικάτος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2020

Ιάσων Μαρμάνης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Ιάσων Μαρμάνης, 2020.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Η εικονική μνήμη αποτελεί την θεμελιώδη αφαίρεση για τον ασφαλή διαμοιρασμό της φυσικής μνήμης ενός υπολογιστικού συστήματος. Με την εικονική μνήμη οι εφαρμογές χρησιμοποιούν εικονικές διευθύνσεις για να αναφερθούν στην μνήμη και το λειτουργικό είναι υπεύθυνο για την παροχή της μετάφρασης από τον εικονικό στον φυσικό χώρο διευθύνσεων. Για την υλοποίηση της συνήθως χρησιμοποιείται ο μηχανισμός της σελιδοποίησης, με τον οποίο το λειτουργικό σύστημα διατηρεί πολυεπίπεδες δομές που περιέχουν τις μεταφράσεις και ονομάζονται Πίνακες Σελίδων (Page Tables). Αυτοί χρησιμοποιούνται κατά την πρόσβαση από το υλικό (Page Walk) ώστε να γίνει η μετάφραση της εικονική διεύθυνσης και να αποθηκευτεί σε κρυφές μνήμες (Translation Lookaside Buffer - TLB) για την επόμενη φορά που θα χρησιμοποιηθεί.

Στην περίπτωση του εικονικού περιβάλλοντος απαιτείται ο συνδυασμός δύο μεταφράσεων, αυτή του εικονικού μηχανήματος (Guest) και αυτή του μηχανήματος που το φιλοξενεί (Host). Η αρχική προσέγγιση για την επίτευξη της εικονικοποίησης της μνήμης, τα Shadow Page Tables, απαιτούν συχνές παρεμβάσεις του Host, ειδικά στην περίπτωση εφαρμογών που αλλάζουν συχνά την διάταξη της μνήμης τους. Η δεύτερη λύση που προτάθηκε, και στηρίζεται στον συνδυασμό των μεταφράσεων από το ίδιο το υλικό, αν και μειώνει τις παρεμβάσεις που απαιτούνται από τον Host, αυξάνει σημαντικά το πλήθος των προσβάσεων που γίνονται κατά τα Page Walks και έτσι είναι ιδιαίτερα κοστοβόρα για εφαρμογές που προσπελαύνουν μεγαλύτερο χώρο διευθύνσεων από αυτόν που μπορεί να καλύψει το TLB. Ως αποτέλεσμα, καμία από τις δύο υπάρχουσες τεχνικές δεν υπερτερεί έναντι της άλλης για κάθε φόρτο εργασίας.

Για να αντιμετωπίσουμε αυτό το πρόβλημα, έχουμε υλοποιήσει ένα μηχανισμό για τον πυρήνα Linux, χρησιμοποιώντας το KVM module και την εφαρμογή QEMU. Ο μηχανισμός μας επιτρέπει σε διαφορετικά εικονικά μηχανήματα να χρησιμοποιούν διαφορετική μέθοδο σελιδοποίησης, η οποία γίνεται να αλλάξει και κατά την διάρκεια εκτέλεσής τους. Επιπλέον προτείνουμε πολιτικές χρήσης του μηχανισμού ώστε να επιλέγεται η βέλτιστη από τις δύο τεχνικές με βάση τον τρέχοντα φόρτο εργασίας. Η δουλειά μας καταφέρνει να έχει επίδοση κοντά σε αυτή της βέλτιστης στατικής επιλογής μεταξύ των δύο μεθόδων, και σε μερικές περιπτώσεις να την ξεπεράσει.

Λέξεις κλειδιά

Εικονική μνήμη, Εικονικοποίηση, Σελιδοποίηση, TLB, KVM

Abstract

Virtual memory is a fundamental abstraction for a computer system, allowing the safe distribution of its memory to the applications. With virtual memory, applications use virtual addresses to access the memory and the Operating System (OS) is responsible for providing the translation from the virtual to the physical address space. The most common mechanism to implement virtual memory, paging, allows the OS to define these mappings using multi-level tree structures, called Page Tables. When a memory address is used, the hardware accesses the Page Tables in order to retrieve the translation, which is also saved inside dedicated caches (Translation Lookaside Buffer - TLB) for future use.

In the virtualized case, the combination of two different mappings is required, the first being used by the virtual machine's OS, and the second used by the host machine's OS. The initial approach to virtualize memory, Shadow Page Tables, requires constant Host intervention, especially in the case of applications that frequently modify their virtual memory layout. The second solution that was proposed, and is based on putting the burden of combining the two mappings on the hardware, even though it almost eliminates the need for Host intervention, it greatly magnifies the Page Walk latency, and thus impacts the performance of applications that address a bigger part of the virtual memory address space than that which can by reached by the TLB. As a result, neither technique is optimal for every workload.

To address this problem, we have implemented a mechanism for the Linux kernel, using the KVM module and QEMU. Our mechanism enables having multiple virtual machines that use different techniques to virtualize memory and allows switching between the two techniques during a virtual machine's runtime. Additionally, we propose two policies in order for the best technique to be determined and selected based on the current workload. Our work can almost match the performance of the best static choice, and in some cases outperform it.

Key words

Virtual Memory, Virtualization, Paging, TLB, KVM

Ευχαριστίες

Ολοκληρώνοντας τις σπουδές μου, θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που με βοήθησαν κατά την διάρκειά τους. Αρχικά θα ήθελα να ευχαριστήσω θερμά τα μέλη του CSLab Χλόη Αλβέρτη, Βασίλειο Καρακώστα και Στράτο Ψωμαδάκη για την πολύτιμη βοήθεια στη παρούσα εργασία, χωρίς τους οποίους δεν θα μπορούσε να έχει έρθει εις πέρας. Επίσης θα ήθελα να ευχαριστήσω τον κύριο Γκούμα για την καθοδήγησή του στην διάρκεια της διπλωματικής αυτής, όπως επίσης και όλους τους καθηγητές του ΕΜΠ που με έκαναν να αγαπήσω το πεδίο της Επιστήμης Υπολογιστών. Τέλος θα ήθελα να ευχαριστήσω τους φίλους μου και την οικογένειά μου για την συνεχή στήριξή τους όλα αυτά τα χρόνια.

Ιάσων Μαρμάνης,

Αθήνα, 2η Ιουλίου 2020

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Εικονική μνήμη και εικονικοποίηση

1.1.1 Εικονική μνήμη

Η εικονική μνήμη είναι η βασική αφαίρεση που επιτρέπει την διαχείριση και τον ασφαλή διαμοιρασμό της φυσικής μνήμης ενός υπολογιστικού συστήματος. Με την εικονική μνήμη κάθε διεργασία διαθέτει τον δικό της συνεχή χώρο εικονικών διευθύνσεων που χρησιμοποιεί για να προσπελάσει τη μνήμη. Το λειτουργικό σύστημα διαχειρίζεται τις αντιστοιχίες των εικονικών διευθύνσεων σε φυσικές μέσω δομών που διατηρεί και ονομάζονται πίνακες σελίδων (Page Tables), ενώ το υλικό είναι υπεύθυνο για την μετάφραση κατά την χρήση της διεύθυνσης προσπελαύνοντας τους πίνακες σελίδων, μια διαδικασία που ονομάζεται περπάτημα σελίδας (Page Walk). Ο μηχανισμός αυτός για την μετάφραση των διευθύνσεων ονομάζεται σελιδοποίηση (paging).

Για την επιτάχυνση της μετάφρασης αυτής, που συνήθως περιλαμβάνει την προσπέλαση μιας πολυεπίπεδης δομής και οδηγεί σε αρκετές προσβάσεις στη μνήμη, οι επεξεργαστές διατηρούν κρυφές μνήμες (Translation Lookaside Buffer - TLB) που αποθηκεύουν τις πιο πρόσφατες μεταφράσεις. Με αυτόν τον τρόπο η προσπέλαση των Page Tables γίνεται μόνο όταν η μετάφραση δεν υπάρχει στο TLB (TLB miss).

1.1.2 Εικονικοποίηση

Η εικονικοποίηση είναι μια τεχνική που επιτρέπει την εκτέλεση πολλαπλών εικονικών μηχανημάτων (Virtual Machine - VM) στο ίδιο μηχάνημα (Host Machine). Τα εικονικά μηχανήματα μπορούν να εκτελούν τον δικό τους πυρήνα και τους δίνεται η ψευδαίσθηση ότι έχουν πλήρη πρόσβαση στους υπολογιστικούς πόρους, η λειτουργία των οποίων προσομοιώνεται από τον Host.

Ένας τρόπος αποδοτικής εικονικοποίησης είναι η τεχνική trap and emulate, με την οποία το εικονικό μηχάνημα εκτελείται απευθείας στο υλικό αλλά όλες οι ευαίσθητες λειτουργίες που απαιτούν την πραγματική πρόσβαση στους υπολογιστικούς πόρους οδηγούν σε εξαίρεση (trap) ώστε να τις χειριστεί κατάλληλα ο Host. Η εικονικοποίηση της αρχιτεκτονικής x86 απαιτεί ειδική υποστήριξη από το υλικό, η οποία και προσφέρεται.

1.1.3 Εικονικοποίηση της μνήμης

Η εικονικοποίηση της μνήμης ενός μηχανήματος απαιτεί τον συνδυασμό δύο διαφορετικών μεταφράσεων, αυτών που διατηρεί το λειτουργικό του εικονικού μηχανήματος (Guest OS) για τις εφαρμογές του και αυτών που διατηρεί το λειτουργικό του Host για το εικονικό μηχάνημα.

Για να επιτευχθεί αυτός ο συνδυασμός υπάρχουν δύο διαφορετικές τεχνικές που χρησιμοποιούνται, τα Shadow Page Tables και η Σελιδοποίηση Δύο Διαστάσεων (Two Dimensional Paging). Με τα Shadow Page Tables το Host λειτουργικό κατασκευάζει και διατηρεί την συνολική μετάφραση, που αποτελείται από την σύνθεση των δύο επιμέρους μεταφράσεων, και το υλικό την χρησιμοποιεί κατά την εκτέλεση του εικονικού μηχανήματος.

Δυστυχώς έτσι απαιτούνται συχνές παρεμβάσεις του Host λειτουργικού στην εκτέλεση του Guest ώστε να μπορέσει να διατηρήσει συγχρονισμένες τις μεταφράσεις. Για αυτόν τον λόγο έχει υλοποιηθεί μια δεύτερη τεχνική που αφήνει τα δύο λειτουργικά να διατηρούν ανεξάρτητα τις δύο μεταφράσεις και αναλαμβάνει το υλικό να τις συνδυάσει κατά την προσπέλαση μίας διεύθυνσης. Η υλοποίηση αυτής της τεχνικής από την Intel ονομάζεται Extended Page Tables (EPT), ενώ από την AMD ως Nested Page Tables (NPT). Στην εργασία αυτή θα χρησιμοποιούμε τον ουδέτερο όρο Two Dimensional Paging (TDP).

Με το TDP, ενώ αποφεύγονται οι παρεμβάσεις του Host που προσθέτει η τεχνική του shadow paging, αυξάνεται η διάρκεια των Page Walks αφού το υλικό πρέπει προσπελάσει ταυτόχρονα και τις δύο δομές με τρόπο που οδηγεί στην πλήρη προσπέλαση των δομών του Host για κάθε επίπεδο προσπέλασης των δομών του Guest OS. Έτσι εφαρμογές που προσπελαύνουν μέρος του χώρου διευθύνσεων μεγαλύτερο από αυτό που μπορεί να καλύψει το TLB υποφέρουν σε ότι αφορά την επίδοση από την χρήση του TDP, το οποίο όμως μπορεί να βελτιώσει την επίδοση σε εφαρμογές που αλλάζουν συχνά την διάταξη της μνήμης τους αφού σε αυτή την περίπτωση το Shadow Paging εισάγει ένα σημαντικό κόστος.

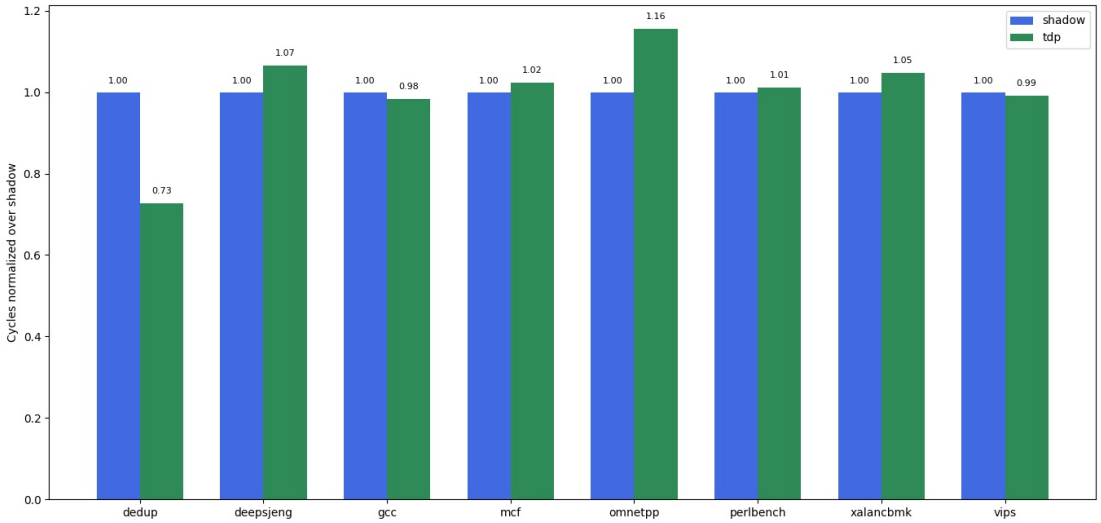
1.2 Εργασία

1.2.1 Κίνητρο

Προηγούμενες δουλειές έχουν δείξει ότι το κόστος της εικονικής μνήμης, και ειδικά στην περίπτωση του εικονικοποιημένου περιβάλλοντος, δεν είναι αμελητέο [Hoan10, Yani16] και πως καμία από τις δύο υπάρχουσες τεχνικές δεν είναι βέλτιστη για κάθε φόρτο εργασίας [Gand16, Kuan14, Bae11, Zhan17]. Οι προτάσεις της μείωση του κόστους αυτού περιλαμβάνουν την εναλλαγή μεταξύ των δύο υπαρχουσών μεθόδων [Kuan14, Bae11, Zhan17], την ταυτόχρονη χρήση τους [Gand16] και την αλλαγή της δομή των Page Tables [Hoan10, Yani16].

Για να επιβεβαιώσουμε τα προηγούμενα αποτελέσματα εκτελέσαμε ένα πλήθος από benchmarks από τις σουίτες μετροπρογραμμάτων Spec2017 και PARSEC. Όπως φαίνεται και στο Σχήμα 1.1 πράγματι καμία από τις δυο τεχνικές δεν είναι καλύτερη σε κάθε περίπτωση, με τις διαφορές να κυμαίνονται από αρκετά μικρές έως και αρκετά σημαντικές.

Σχήμα 1.1: Κύκλοι εκτέλεσης για διάφορα benchmarks, κανονικοποιημένοι ως προς το shadow paging



1.2.2 Μηχανισμός και πολιτικές

Για να ξεπεράσουμε το πρόβλημα αυτό, υλοποιήσαμε αρχικά έναν μηχανισμό που μας επιτρέπει να χρησιμοποιούμε διαφορετικές μεθόδους εικονικοποίησης της μνήμης για διαφορετικά εικονικά μηχανήματα του ίδιου Host, όπως επίσης και να αλλάζουμε την μέθοδο που χρησιμοποιείται κατά τον χρόνο εκτέλεσης του εικονικού μηχανήματος.

Προτείνουμε και αξιολογούμε δύο πολιτικές που μπορούν να χρησιμοποιήσουν τον μηχανισμό ώστε να εντοπίζεται και να επιλέγεται η βέλτιστη μέθοδος, με στόχο την μείωση του συνολικού κόστους της εικονικοποίησης της μνήμης. Και οι δύο πολιτικές βασίζονται στην παρακολούθηση του ποσοστού των κύκλων που αφιερώνονται από το εικονικό μηχάνημα σε Page Walks και σε παρεμβάσεις από τον Host, και η απόφαση για την αλλαγή λαμβάνεται με βάση κάποια κατώφλια.

Η πρώτη πολιτική χρησιμοποιεί σταθερές προκαθορισμένες τιμές για τα κατώφλια αυτά. Επειδή παρατηρήσαμε πως διαφορετικές εφαρμογές έχουν διαφορετικές τιμές κατωφλιών που οδηγούν στην βέλτιστη απόδοση του μηχανισμού μας, δοκιμάσαμε και αξιολογήσαμε μία δεύτερη πολιτική με την οποία τα κατώφλια αλλάζουν δυναμικά με βάση την αλλαγή στην επίδοση του εικονικού μηχανήματος που παρατηρείται μετά την αλλαγή των μεθόδων. Συνολικά, η δουλειά μας καταφέρνει να έχει επίδοση κοντά σε αυτή της βέλτιστης στατικής επιλογής μεταξύ των δύο μεθόδων, και σε μερικές περιπτώσεις να την ξεπεράσει.

Στην δουλειά αυτή έχουμε εστιάσει στον πυρήνα Linux και το KVM module, το οποίο χρησιμοποιείται μαζί με την εφαρμογή QEMU για την εκτέλεση εικονικών μηχανημάτων. Παρόμοιος μηχανισμός έχει υλοποιηθεί για το KVM, ο οποίος όμως δεν επιτρέπει την εκτέλεση παραπάνω από ενός εικονικού μηχανήματος στο ίδιο μηχάνημα. Εμείς ακολουθήσαμε μια διαφορετική προσέγγιση, αποσυνδέοντας πρώτα την μέθοδο σελιδοποίησης

από το global state του KVM, και μετά υλοποιήσαμε την δυνατότητα αλλαγής μεθόδου. Το σύνολο του κώδικά μας για τον μηχανισμό και τις πολιτικές μπορεί να βρεθεί στο https://github.com/jmarmanis/ntua-rpms.

1.2.3 Περίγραμμα

Αρχικά, παρουσιάζουμε στο Κεφάλαιο 2 το απαραίτητο υπόβαθρο για την κατανόηση της εργασίας. Στο Κεφάλαιο 3 περιγράφουμε την υλοποίηση του μηχανισμού μας και των δύο πολιτικών που εξετάσαμε. Στο Κεφάλαιο 4 αξιολογούμε τις πολιτικές, παρουσιάζοντας την μεθοδολογία και τα αποτελέσματα των πειραμάτων που εκτελέσαμε. Τέλος, στο Κεφάλαιο 5 αναφέρουμε τις σχετικές δουλειές που έχουν γίνει, και στο Κεφάλαιο 6 συνοψίζουμε και προτείνουμε πιθανές επεκτάσεις.

Κεφάλαιο 2

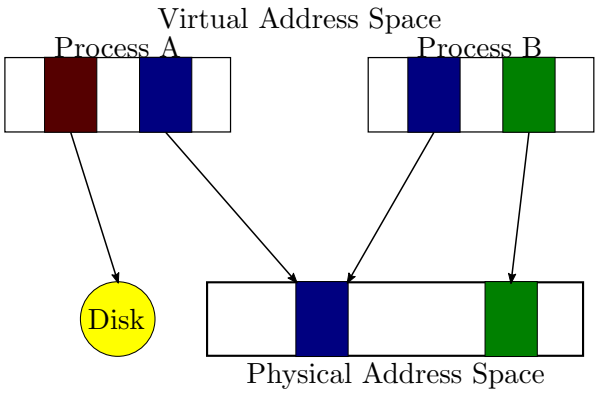
Θεωρητικό υπόβαθρο

2.1 Εικονική Μνήμη

Το λειτουργικό σύστημα αποτελεί μια διεπαφή μεταξύ του υλικού και των εφαρμογών και είναι υπεύθυνο για την διαχείριση των πόρων του συστήματος. Για την διαχείριση της φυσικής μνήμης το λειτουργικό σύστημα χρησιμοποιεί την αφαίρεση της εικονικής μνήμης.

Κάθε πρόγραμμα αντί να αναφέρεται απευθείας στην φυσική μνήμη, χρησιμοποιεί διευθύνσεις που ανήκουν σε έναν εικονικό χώρο διευθύνσεων και διαμοιράζονται από το λειτουργικό, που είναι υπεύθυνο για την αντιστοίχιση μεταξύ των εικονικών και των φυσικών διευθύνσεων. Γενικά η αντιστοίχιση είναι διαφορετική ανά διεργασία. Μία έγκυρη εικονική διεύθυνση δεν αντιστοιχεί απαραίτητα σε φυσική μνήμη αφού μπορεί τα δεδομένα να μην βρίσκονται στην κύρια μνήμη. Στο Σχήμα 2.1 φαίνεται ένα παράδειγμα του μηχανισμού αυτού.

Σχήμα 2.1: Εικονικός χώρος διευθύνσεων 2 διεργασιών. Μέρος των δεδομένων δεν βρίσκονται στη μνήμη.

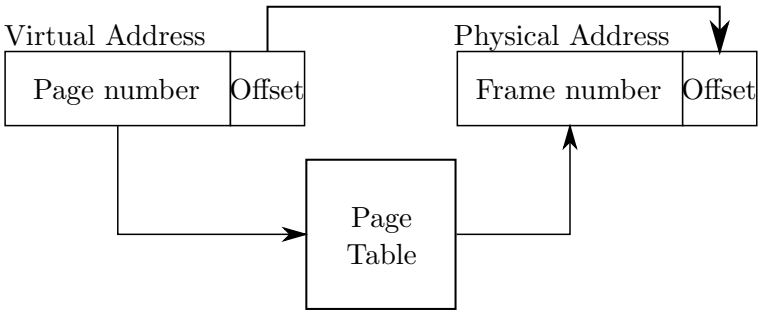


2.1.1 Σελιδοποίηση

Η πιο συνηθισμένη υλοποίηση της εικονικής μνήμης χρησιμοποιεί τον μηχανισμό της σελιδοποίησης (paging). Ο φυσικός και ο εικονικός χώρος διευθύνσεων χωρίζεται σε τμήματα σταθερού και ίσου μήκους. Τα τμήματα της φυσικής και της εικονικής μνήμης ονομάζονται

πλαίσια (frames) και σελίδες (pages), αντίστοιχα, και το κοινό τους μήκος ονομάζεται μέγεθος σελίδας. Κάθε αναφορά σε μία διεύθυνση ερμηνεύεται ως ένας αριθμός σελίδας μαζί με μία μετατόπιση, όπως φαίνεται στο Σχήμα 2.2. Το λειτουργικό παρέχει τις μεταφράσεις από πλαίσια σε σελίδες μέσω κάποιων δομών που ονομάζονται πίνακες σελίδων (page tables). Η αντιστοίχιση αυτή μαζί με την μετατόπιση ορίζει την μετάφραση από εικονικές σε φυσικές διευθύνσεις.

Σχήμα 2.2: Μετάφραση εικονικής διεύθυνσης μέσω του μηχανισμού της σελιδοποίησης



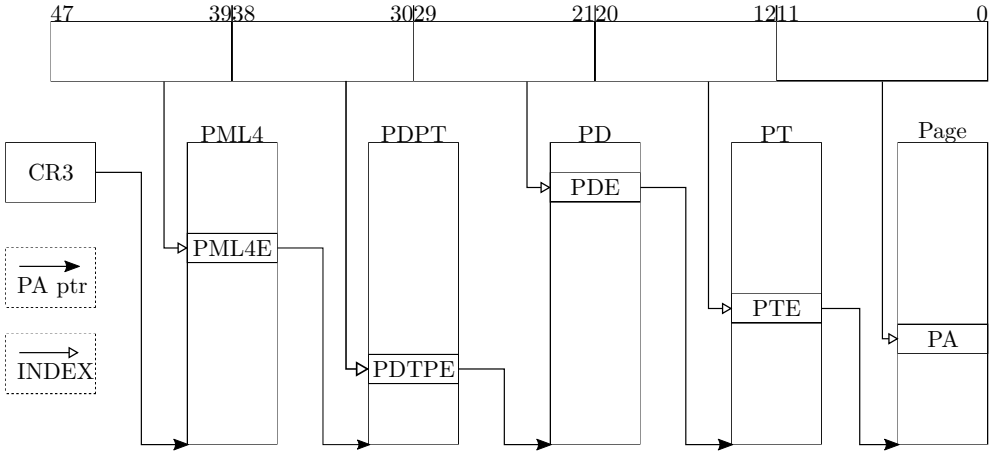
2.1.2 Πίνακες σελίδων

Οι πίνακες σελίδων αποτελούν τις δομές που χρησιμοποιεί το λειτουργικό, ώστε να μεταφράσει μια εικονική διεύθυνση σε φυσική. Μια τυπική υλοποίηση τους, που χρησιμοποιείται και από την αρχιτεκτονική x86-64, είναι αυτή της πολυεπίπεδης δενδρικής δομής. Η δομή αυτή αποτελείται από ένα σύνολο από πίνακες, με κάθε ένα να βρίσκεται σε ένα ξεχωριστό πλαίσιο, και κάθε εγγραφή ενός πίνακα περιλαμβάνει έναν αριθμό πλαισίου μαζί με κάποιες επιπλέον πληροφορίες, όπως το αν το πλαίσιο αυτό είναι έγκυρο, αν έχει γίνει πρόσβαση ή εγγραφή σε αυτό, και ποια είναι τα δικαιώματα πρόσβασης. Για να βρεθεί η φυσική διεύθυνση που αντιστοιχεί σε μία εικονική βάσει ενός συνόλου από page tables, η εικονική διεύθυνση χωρίζεται σε τμήματα που χρησιμοποιούνται για να επιλεγεί η καταχώριση του πίνακα που οδηγεί στον πίνακα του επόμενου επιπέδου, κοκ. Η φυσική διεύθυνση προκύπτει άμεσα από τον τελικό αριθμό πλαισίου μαζί με την μετατόπιση. Η διαδικασία αυτή ονομάζεται περπάτημα σελίδας (Page Walk).

Για λόγους εξοικονόμησης χώρου, το λειτουργικό δεν αποθηκεύει την αναπαράσταση όλου του δέντρου αλλά μόνο των τμημάτων του δέντρου που καταλήγουν σε έγκυρες μεταφράσεις. Κάθε σύνολο από πίνακες σελίδων αντιστοιχεί σε ένα ξεχωριστό χώρο διευθύνσεων και συνήθως αφορά μία συγκεκριμένη διεργασία.

Στην αρχιτεκτονική x86-64 το δέντρο αποτελείται από 4 επίπεδα και το μέγεθος σελίδας είναι 4KiB. Κάθε καταχώρηση του πίνακα σελίδων έχει μέγεθος 8 bytes άρα προκύπτουν 512 καταχωρήσεις ανά πίνακα σελίδων για τις οποίες αρκούν 9 bits για να δεικτοδοτηθούν. Η (φυσική) διεύθυνση του ριζικού πίνακα σελίδων είναι αποθηκευμένη σε ένα καταχωρητή ελέγχου, τον CR3. Στο Σχήμα 2.3 φαίνεται η λειτουργία του page walk.

Σχήμα 2.3: Περπάτημα πινάκων σελίδων 4 επιπέδων για την αρχιτεκτονική x86-64



2.1.3 Σφάλμα σελίδας

Αν κατά το περπάτημα σελίδας μια μετάφραση δεν υπάρχει ή υπάρχει αλλά τα δικαιώματα πρόσβασης δεν επιτρέπουν την πρόσβασή της τότε το υλικό προκαλεί ένα σφάλμα σελίδας ώστε το λειτουργικό να το χειριστεί, πιθανώς φέρνοντας την ζητούμενη σελίδα από το δίσκο και ανανεώνοντας τους πίνακες σελίδων.

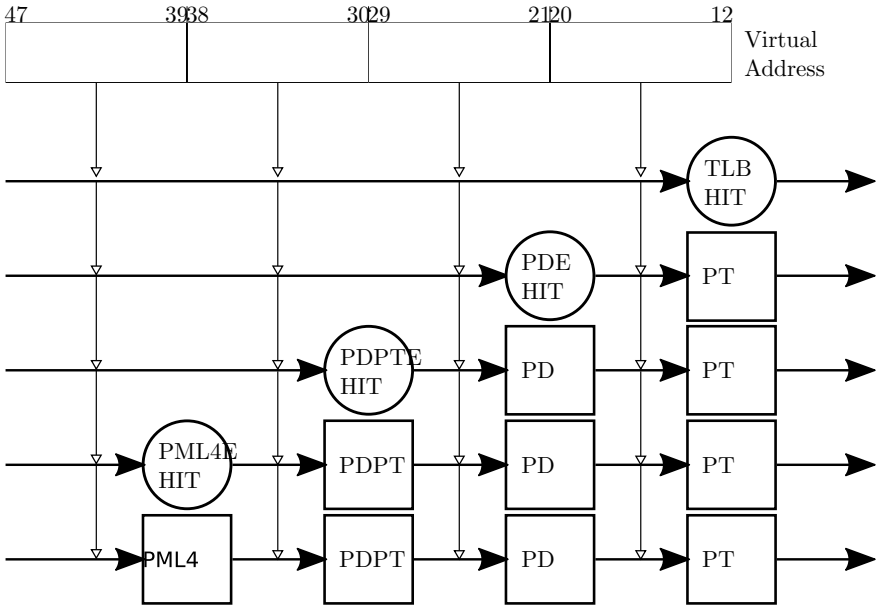
Αυτός ο μηχανισμός χρησιμοποιείται εκτεταμένα από το λειτουργικό σύστημα για την διαχείριση της μνήμης. Συγκεκριμένα όταν μια εφαρμογή ζητά ένα εύρος διευθύνσεων για χρήση, το λειτουργικό δεν αλλάζει τους πίνακες σελίδων της παρά μόνο όταν η εφαρμογή κάνει πρόσβαση σε μία από αυτές τις διευθύνσεις και οδηγήσει σε σφάλμα σελίδων.

2.1.4 TLB και Paging Structure Caches

Με τον μηχανισμό της σελιδοποίησης κάθε πρόσβαση στη μνήμη από την εφαρμογή ή τον πυρήνα θα οδηγούσε σε ένα page walk ώστε να βρεθεί η αντίστοιχη φυσική διεύθυνση. Επειδή αυτό θα ήταν εξαιρετικά κοστοβόρο, το υλικό διαθέτει μια κρυφή μνήμη (cache) που ονομάζεται Translation Lookaside Buffer (TLB) και αποθηκεύει τις μεταφράσεις ώστε να χρησιμοποιηθούν ξανά.

Αφού κάθε αλλαγή στα page tables δεν γίνεται αντιληπτή από το υλικό, όταν αλλάζει μια μετάφραση πρέπει να ακυρώνονται οι - πλέον μη έγκυρες - αποθηκευμένες μεταφράσεις. Για να γίνει αυτό το υλικό προσφέρει τις κατάλληλες εντολές (π.χ. INVLPG στην x86-64). Επειδή κάθε επεξεργαστής διαθέτει το δικό του TLB, κάθε φορά που αλλάζουν οι μεταφράσεις για έναν επεξεργαστή είναι πιθανό να πρέπει να ενημερωθούν και οι υπόλοιποι ώστε να ακυρώσουν και αυτοί τις μεταφράσεις τους. Η διαδικασία αυτή ονομάζεται TLB Shootdown.

Σχήμα 2.4: Λειτουργία των Paging Structure Caches



Για να επιταχύνει περαιτέρω το page walk το υλικό χρησιμοποιεί και δευτερεύουσες caches που αποθηκεύουν καταχωρήσεις των page tables. Συγκεκριμένα, στην υλοποίηση της Intel για την αρχιτεκτονική x86-64 οι δομές αποθηκεύουν μερικώς τις μεταφράσεις, επιτρέποντας να παραληφθούν κάποια επίπεδα κατά την διάσχιση των page tables, ενώ στης AMD αποθηκεύονται οι εγγραφές των page tables (page table entries - pte) βάσει της φυσικής τους διεύθυνσης ώστε να γίνεται πιο γρήγορα η πρόσβαση σε κάθε pte. Η μονάδα που διαχειρίζεται κάθε αναφορά στη μνήμη και είναι υπεύθυνη για την μετάφραση των διευθύνσεων ονομάζεται MMU.

2.2 Εικονικοποίηση

2.2.1 Ορισμοί

Εικονικοποίηση υλικού ονομάζεται η τεχνική με την οποία προσομοιώνουμε την ύπαρξη εικονικών μηχανημάτων μέσα σε ένα μηχάνημα. Το εικονικό μηχάνημα ονομάζεται και guest μηχάνημα, ενώ το μηχάνημα στο οποίο τρέχει ονομάζεται host μηχάνημα. Το λογισμικό που υλοποιεί το εικονικό υλικό ονομάζεται Virtual Machine Monitor.

Το εικονικό μηχάνημα έχει την ψευδαίσθηση ότι υπάρχουν πραγματικοί πόροι όπως ο επεξεργαστής, η μνήμη και οι συσκευές στους οποίους έχει πλήρη πρόσβαση. Στην πραγματικότητα, το εικονικό μηχάνημα χρησιμοποιεί πόρους που είτε δανείζονται ή προσομοιώνονται από το host μηχάνημα.

2.2.2 Υποστήριξη υλικού

Ένας τρόπος αποδοτικής εικονικοποίησης είναι η τεχνική trap and emulate. Το εικονικό μηχάνημα τρέχει απευθείας στο υλικό αλλά σε χαμηλό τρέχον επίπεδο προνομίων (Current Privilege Level). Κάθε εκτέλεση εντολής που αφορά διαχείριση πόρων του συστήματος θα προκαλεί μια εξαίρεση υλικού, αφήνοντας το host μηχάνημα να την προσομοιώσει, ενώ οι υπόλοιπες εκτελούνται άμεσα από τον επεξεργαστή.

Αυτή η τεχνική δεν μπορεί να εφαρμοστεί στην περίπτωση της αρχιτεκτονικής x86-64 επειδή μερικές εντολές αντί να προκαλέσουν εξαίρεση συμπεριφέρονται διαφορετικά όταν εκτελούνται σε χαμηλότερο CPL. Για να ξεπεραστεί αυτό το πρόβλημα και να μπορεί να εικονικοποιηθεί η αρχιτεκτονική x86-64 το υλικό προσφέρει ειδική υποστήριξη. Παρακάτω θα περιγράψουμε την υλοποίηση της Intel για αυτή την λειτουργία που ονομάζεται Virtual Machine Extensions (VMX) [Inte].

Αφού ο επεξεργαστής ενεργοποιήσει το VMX, πλέον μπορεί να βρίσκεται σε δύο διαφορετικές καταστάσεις: VMX root και VMX non-root. Ο VMM συνήθως τρέχει σε VMX root mode ενώ το guest μηχάνημα σε VMX non-root. Η μετάβαση από VMX root σε non root ονομάζεται VM entry ενώ το αντίστροφο VM exit.

Όταν ο επεξεργαστής τρέχει σε VMX non-root mode τότε περιορίζεται η λειτουργία του, με κάποια γεγονότα και εντολές να προκαλούν VM exits. Αυτό αφήνει τον VMM να διαχειριστεί το αντίστοιχο γεγονός ή εντολή.

Ο έλεγχος των μεταβάσεων, δηλαδή πότε και τι συμβαίνει κατά τις μεταβάσεις VMX, γίνεται μέσω μίας δομής που ονομάζεται VMCS, στην οποία το υλικό έχει πρόσβαση μέσω

ενός δείκτη που θέτει ο VMM. Σε κάθε εικονικό μηχάνημα, ή σε κάθε εικονικό επεξεργαστή αν το εικονικό μηχάνημα διαθέτει πολλαπλούς επεξεργαστές, αντιστοιχεί ένα VMCS.

2.2.3 KVM

Στο Linux, η χρήση των VMX για την εκτέλεση εικονικών μηχανημάτων προσφέρεται μέσω ενός module που ονομάζεται KVM [Qumr07]. Αφού το module φορτωθεί, ο πυρήνας μπορεί να λειτουργήσει και ως VMM.

Η λειτουργικότητα αυτή προσφέρεται στον χώρο χρήση μέσω ενός ειδικού αρχείου (/dev/kvm). Ανοίγοντας το αρχείο αυτό και με εκτέλεση κλήσεων ioctl μία διεργασία έχει την δυνατότητα να δημιουργήσει, να τρέξει και να διαχειριστεί μία εικονική μηχανή.

Η διεργασία του χώρου χρήστη είναι υπεύθυνη για να προσομοιώσει κάθε λειτουργία ή συσκευή του εικονικού μηχανήματος που δεν μπορεί να εξυπηρετηθεί από τον VMM. Η πιο συνηθισμένη διεργασία χώρου χρήστη είναι το QEMU και έτσι θα χρησιμοποιούμε αυτή ως παράδειγμα.

2.3 Εικονικοποίηση της μνήμης

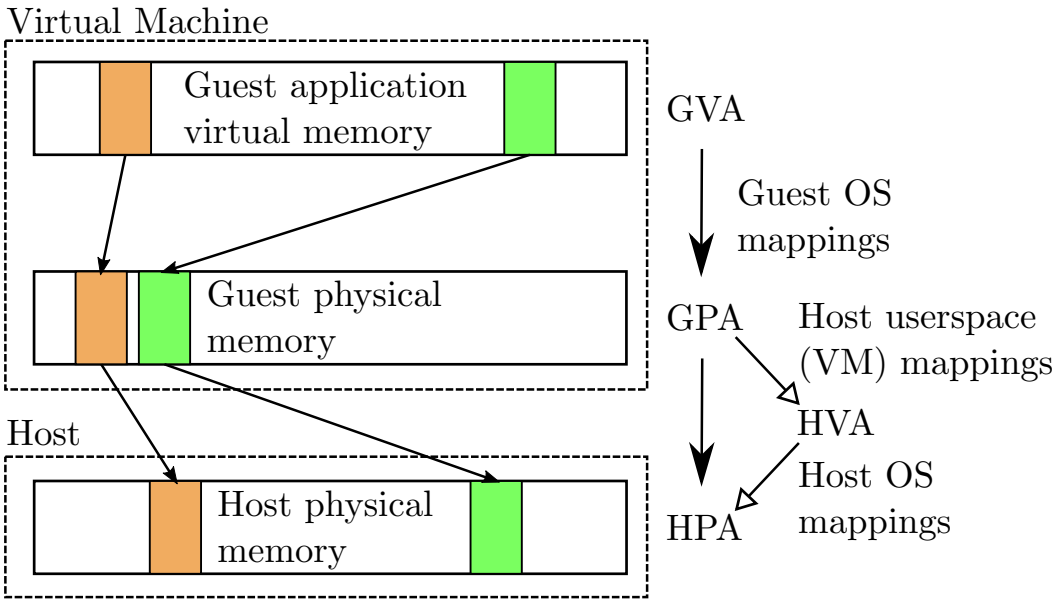
2.3.1 Μεταφράσεις διευθύνσεων

Όπως και με κάθε πόρο του μηχανήματος, στο εικονικό μηχάνημα δεν δίνεται άμεση πρόσβαση στη μνήμη. Για να μπορεί να την χρησιμοποιήσει, πρέπει το QEMU να του διαθέσει ένα μέρος της δικής του μνήμης. Συγκεκριμένα, το QEMU καταχωρεί κάποια τμήματα της εικονικής του μνήμης ως διαθέσιμα για το εικονικό μηχάνημα, ορίζοντας παράλληλα και την αντιστοιχία μεταξύ των φυσικών διευθύνσεων του guest (Guest Physical Address, GPA), και των εικονικών διευθύνσεών του (Host Virtual Address, HVA). Κάθε τέτοιο τμήμα αναφέρεται ως Memory Slot. Η αντιστοίχιση από GPA σε φυσικές διευθύνσεις του host (Host Physical Address) προκύπτει συνδυάζοντας την αντιστοίχιση GPA $\rightarrow$ HVA με τις μεταφράσεις HVA $\rightarrow$ HPA της διεργασίας QEMU.

Όταν ο Guest έχει ενεργοποιήσει την σελιδοποίηση οι διευθύνσεις του πρέπει να περάσουν από 2 επίπεδα μετάφρασης ώστε να προσπελαστεί η μνήμη, από εικονικές διευθύνσεις του guest (Guest Virtual Address, GVA) σε GPA και από GPA σε HPA.

Συνολικά λοιπόν έχουμε τις παρακάτω μεταφράσεις, όπως συνοψίζονται και στο Σχήμα 2.5:

- Host kernel για την QEMU διεργασία: HVA $\rightarrow$ HPA
- QEMU διεργασία για το VM (Memory Slots): GPA $\rightarrow$ HVA
- GPA $\rightarrow$ HVA $\circ$ HVA $\rightarrow$ HPA: GPA $\rightarrow$ HPA
- Guest kernel για guest διεργασία: GVA $\rightarrow$ GPA
- GVA $\rightarrow$ GPA $\circ$ GPA $\rightarrow$ HPA: GVA $\rightarrow$ HPA



Σχήμα 2.5: Επίπεδα μεταφράσεων σε εικονικοποιημένο περιβάλλον.

Στην περίπτωση που ο guest δεν έχει ενεργοποιήσει την σελιδοποίηση τότε δεν υπάρχει το επίπεδο μετάφρασης GVA → GPA και ο Guest χρησιμοποιεί άμεσα GPAs.

Για να εικονικοποιηθεί η μνήμη ο VMM υλοποιεί ένα εικονικό MMU που είναι υπεύθυνο να κάνει τις απαραίτητες μεταφράσεις.

2.3.2 Shadow Page Tables

Η πρώτη μέθοδος για την εικονικοποίηση του MMU είναι αυτή των Shadow Page Tables. Με αυτή την τεχνική, το KVM εκτελεί πάντα το VM με ενεργοποιημένη την σελιδοποίηση και κατασκευάζει τους σκιώδεις Πίνακες Σελίδων (Shadow Page Tables) που ορίζουν πως μεταφράζονται οι διευθύνσεις που χρησιμοποιεί ο Guest σε φυσικές διευθύνσεις. Το υλικό χρησιμοποιεί αποκλειστικά αυτούς τους πίνακες για την μετάφραση διευθύνσεων.

Στην περίπτωση που ο Guest έχει ενεργοποιήσει την σελιδοποίηση τα shadow page tables κρατάνε την συνολική μετάφραση GVA → HPA. Για να γίνει αυτό, για κάθε page table του Guest OS το KVM έχει ένα Shadow Page Table και προσπαθεί να κρατήσει αυτά τα δύο συγχρονισμένα. Η σχέση μεταξύ των δύο συνόλων από page tables φαίνεται στο Σχήμα 2.6.

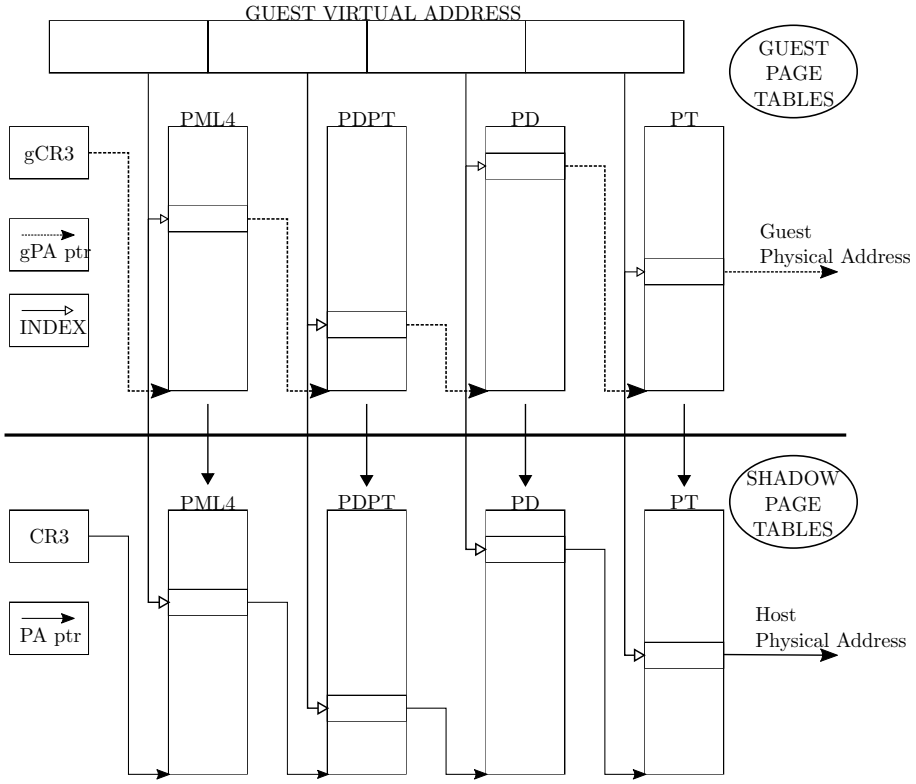
Το KVM ρυθμίζει το VMCS ώστε κάθε εντολή που αφορά το MMU να προκαλέσει VM Exit ώστε να προσομοιωθεί η απαραίτητη λειτουργία. Τα shadow page tables φτιάχνονται οκνηρά κατά τα page faults του guest, συνδυάζοντας τις μεταφράσεις GVA → GPA του Guest και GPA → HPA του VMM.

Στην αρχική υλοποίηση, το KVM κρατούσε συγχρονισμένα τα δύο σύνολα από page tables μόνο μέσω εντολών που ακύρωναν τις αποθηκευμένες μεταφράσεις (όπως INVLPG) ή που άλλαζαν το ριζικό page table (εγγραφή στον CR3). Μετά τα αρχικά page faults η

διεργασία του guest θα μπορούσε να τρέχει σε κανονική ταχύτητα. Αυτό όμως οδηγούσε στο να κατασκευαστούν ξανά οι πίνακες σελίδων σε κάθε αλλαγή τρέχουσας διεργασίας (context switch)του guest.

Για να μπορούν να αποθηκεύονται τα page tables μεταξύ των guest context switches το KVM παρακολουθεί κάθε εγγραφή που γίνεται στα Page Tables του Guest, αφαιρώντας το δικαίωμα εγγραφής σε σελίδες που αντιστοιχούν σε Page Tables. Έτσι κάθε αλλαγή στα Page Tables προκαλεί VM Exits ώστε ο VMM να συγχρονίσει τα δικά του αντίγραφα. Η αφαίρεση του δικαιώματος εγγραφής στα Guest Page Tables απαιτεί από τον VMM να γνωρίζει για κάθε πλαίσιο του guest ποιά shadow page table entries (spte) το αφορούν. Για την λειτουργία αυτή υπάρχουν οι reverse map δομές, οι οποίες δίνουν τα sptes που αντιστοιχούν σε ένα αριθμό πλαισίου του guest (Guest Frame Number, GFN).

Σχήμα 2.6: Shadow Page Tables



Μια βελτιστοποίηση μπορεί να εφαρμοστεί όταν ένα Guest Page Tables ανήκει στα Page Tables που ξεκινάνε από τον CR3 που βρίσκεται σε χρήση από τον Guest. Μια αλλαγή σε ένα τέτοιο Page Table δεν χρειάζεται να προκαλέσει VM exit αφού για να χρησιμοποιήσει την νέα μετάφραση ο Guest οφείλει να εκτελέσει μία εντολή ακύρωσης των αποθηκευμένων μεταφράσεων (INVLPG). Τότε ο VMM μπορεί να συγχρονίσει όλα τα μη-συγχρονισμένα Page Tables μαζί. Στο KVM, μόνο τα Page Tables του τελευταίου επιπέδου αφήνονται ασυγχρόνιστα.

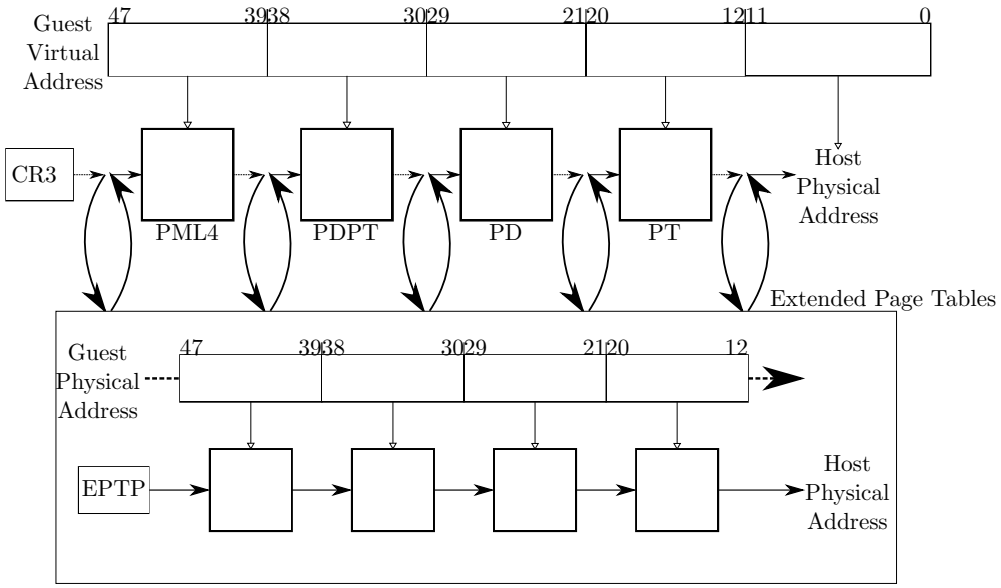
Στην περίπτωση που ο guest δεν έχει ενεργοποιήσει ακόμα την σελιδοποίηση τα Shadow

Page Tables περιέχουν την μετάφραση GPA → HPA.

2.3.3 Two Dimensional Paging

Η δεύτερη τεχνική που χρησιμοποιείται για την εικονικοποίηση της μνήμης απαιτεί την υποστήριξη του υλικού και ονομάζεται Σελιδοποίηση Δύο Διαστάσεων (Two Dimensional Paging). Το υλικό προσφέρει τη δυνατότητα στο VMM να καθορίσει μέσω ενός συνόλου από page tables την μετάφραση GPA → HPA, αφήνοντας τον guest να διαχειρίζεται ο ίδιος τα Page Tables του χωρίς καμία επέμβαση από τον VMM. Όταν χρειαστεί η μετάφραση για μία διεύθυνση του Guest, το υλικό χρησιμοποιεί και τις δύο δομές ώστε να υπολογίσει την τελική μετάφραση. Με αυτό τον τρόπο ούτε οι αλλαγές στα Page Tables, ούτε οι προσβάσεις στον CR3 προκαλούν VM exits, μειώνοντας σημαντικά το κόστος διαχείρισης της μνήμης του Guest από τον Host.

Σχήμα 2.7: Page walk στην περίπτωση των Extended Page Tables



Παρ' όλα αυτά, με αυτήν την τεχνική προστίθεται ένα σημαντικό κόστος κατά την διάρκεια της διάσχισης των Page Tables. Συγκεκριμένα, το υλικό οφείλει κατά το Page Table walk να μετατρέπει κάθε GPA σε HPA για να μπορέσει να προσπελάσει το επόμενο επίπεδο, οδηγώντας σε ένα δισδιάστατο περπάτημα των Page Tables (Two dimensional Page Table Walk) με τις 4 προσβάσεις στη μνήμη που χρειάζονται στην αρχιτεκτονική x86-64 για το Page Table Walk, να αυξάνονται σε 24 στην περίπτωση του Two Dimensinal Paging (TDP). Η διαδικασία αυτή περιγράφεται στο Σχήμα 2.7.

Τα Page Tables που αφορούν τις μεταφράσεις GPA → HPA θα τα αναφέρουμε ως Extended Page Tables (EPT), που είναι και το όνομα που χρησιμοποιεί η Intel για την υλοποίησή της [Inte]. Τα EPTs κατασκευάζονται οκνηρά κατά τις αρχικές προσβάσεις στη φυσική μνήμη του VM από το VMM συνδυάζοντας πάλι τις μεταφράσεις HVA → HPA της

διεργασίας που διαχειρίζεται το VM και τις μεταφράσεις GPA → HVA των memory slots. Η υλοποίηση τους εκμεταλλεύεται όλη την υποδομή που υπήρχε από τα Shadow Page Tables, χρησιμοποιώντας ξανά τις υπάρχουσες δομές, αφού έχει πολλά κοινά με την περίπτωση κατά την οποία το λειτουργικό του guest δεν έχει ενεργοποιήσει την σελιδοποίηση και τα Shadow Page Tables του VMM περιέχουν πάλι μόνο τις μεταφράσεις GPA → HPA.

2.3.4 Σύγκριση και βελτιστοποιήσεις

Και με τις δύο τεχνικές η εικονικοποίηση της μνήμης προσθέτει ένα επιπλέον κόστος στην δουλειά που χρειάζεται για την εικονικοποίηση ενός VM. Από την μία με τα Shadow Page Tables ο host πρέπει να παρακολουθεί τα Page Faults και τις προσβάσεις στα Page Tables και στον CR3 ώστε να διαδώσει τις αλλαγές που γίνονται στις δικές του δομές, προκαλώντας πολλά VM exits. Από την άλλη, με τα Extended Page Tables αυξάνεται σημαντικά το πλήθος των προσβάσεων στη μνήμη στην περίπτωση ενός TLB miss.

Στην πρώτη περίπτωση, το κόστος είναι μεγάλο όταν το φορτίο που εκτελείται μεταβάλει συχνά την διάταξη της μνήμης τους, ενώ στην δεύτερη περίπτωση όταν η εφαρμογή προσπελαύνει ένα εύρος διευθύνσεων μεγαλύτερο από αυτό που μπορεί να καλυφθεί από το TLB (TLB reach), οδηγώντας σε χρονοβόρα 2D Page Table Walks.

Σε αυτή την υποενότητα περιγράφουμε αρχικά κάποιες βελτιστοποιήσεις που μπορούν να εφαρμοστούν για να μειωθεί το μέρος του virtualization overhead που αφορά τη μνήμη και είναι ανεξάρτητες από τις 2 προηγούμενες τεχνικές.

Ύστερα περιγράφουμε δύο τεχνικές που χρησιμοποιούνται για να περιορίσουν τον χρόνο που απαιτείται για την εξυπηρέτηση ενός TLB miss (TLB miss latency) στην περίπτωση του TDP. Η πρώτη το επιτυγχάνει προσπερνώντας κάποια επίπεδα κατά το Page Walk χρησιμοποιώντας caches ενώ η δεύτερη μειώνει το πλήθος των επιπέδων που χρειάζεται να προσπελαστούν κατά το Page Walk αυξάνοντας το μέγεθος της σελίδας. Επιπλέον αναφέρουμε ένα πρόσφατο χαρακτηριστικό που προστέθηκε στην αρχιτεκτονική x86-64 και έχει αρνητική επίπτωση στο TLB miss latency.

Ασύγχρονα Page Faults

Τα ασύγχρονα Page Faults είναι ένας τρόπος ώστε να μειωθεί ο χρόνος που ο Host αναστέλλει την λειτουργία του Guest μηχανήματος λόγω πρόσβασης σε μνήμη που δεν είναι άμεσα διαθέσιμη. Συγκεκριμένα, όταν ο Guest προσπαθήσει να προσπελάσει δεδομένα που δεν βρίσκονται στην φυσική μνήμη, ο Host κανονικά θα πρέπει να κοιμίσει το VM μέχρι τα δεδομένα αυτά να μεταφερθούν στη μνήμη. Με τα ασύγχρονα Page Faults, ο Guest μπορεί να ενημερωθεί για το γεγονός άμεσα ώστε να χρονοδρομολογήσει πιθανώς μια άλλη διεργασία μέχρι τα δεδομένα να έρθουν στη μνήμη από τον host, οπότε και θα ενημερωθεί πάλι για να συνεχίσει την εκτέλεση της αρχικής διεργασίας.

Address Space Identifiers

Το TLB και οι υπόλοιπες Paging Structures caches μοιράζονται μεταξύ των διαφορετικών διεργασιών του λειτουργικού αλλά και μεταξύ του λειτουργικού συστήματος και των

VMs. Αυτό οδηγεί στην ανάγκη για άδειασμα (flush) των δομών αυτών ώστε να μην χρησιμοποιηθούν παλιότερες άκυρες μεταφράσεις. Το υλικό υποστηρίζει την ύπαρξη πολλαπλών address spaces στο TLB και στις υπόλοιπες Paging Structure caches ώστε αυτό να αποφεύγεται. Συγκεκριμένα, όταν γίνεται χρήση τους, κάθε καταχώρηση του TLB συνδέεται με ένα συγκεκριμένο address space και χρησιμοποιείται μόνο αν ταιριάζει με το τρέχον.

Στην αρχιτεκτονική x86, το αναγνωριστικό ενός address space είναι ο συνδυασμός 2 αναγνωριστικών, του Virtual Process Identifier (VPID) και του Process Context Identifier (PCID). Το πρώτο είναι ένα μοναδικό αναγνωριστικό που δίνεται σε κάθε εικονικό επεξεργαστή ενός VM, με την μηδενική τιμή να αντιστοιχεί στον host, ενώ το δεύτερο διαμοιράζεται από το λειτουργικό και συνήθως είναι μοναδικό ανά διεργασία.

Αν και η υποστήριξη από το υλικό υπήρχε και προηγουμένως, τα PCIDs το Linux τα υποστήριξε μόλις πρόσφατα, κυρίως λόγω της υλοποίησης του Kernel Page Table Isolation (kPTI) [LWN17a], ενός γνωρίσματος που ενσωματώθηκε στον πυρήνα ως τρόπος προστασίας από την ευπάθεια ασφαλείας υλικού Meltdown.

Πριν το kPTI, κάθε διεργασία είχε ένα μοναδικό σύνολο από Page Tables που χρησιμοποιούνταν και κατά την μετάβαση μιας διεργασίας σε χώρο πυρήνα. Για να προστατευτεί ο πυρήνας από το Meltdown, δεν χρησιμοποιεί πλέον τα ίδια Page Tables για τον εαυτό του και τον χώρο χρήστη. Ο χώρος χρήστη χρησιμοποιεί ένα σύνολο από Page Tables που αφορά τον χώρο διευθύνσεων της εφαρμογή ενώ ο πυρήνας το πλήρες σύνολο που περιέχει και τις μεταφράσεις που αντιστοιχούν στον χώρο πυρήνα. Επειδή το τρέχον σύνολο από Page Tables αλλάζει σε κάθε μετάβαση από και προς τον πυρήνα, η χρήση των PCIDs είναι απαραίτητη για να μην γίνεται flush των Page Tables σε κάθε κλήση συστήματος, αφού αυτό θα μείωνε σημαντικά την απόδοση του συστήματος.

Cache για Two Dimensional Page Walk

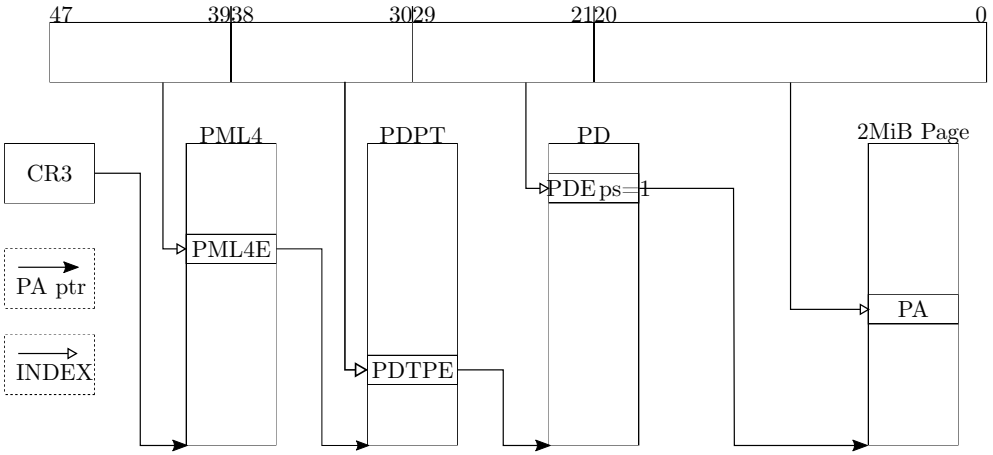
Όπως και στην κανονική λειτουργία του επεξεργαστή, έτσι και κατά την εκτέλεση του όταν τρέχει ένα VM με την υποστήριξη του υλικού (VMX) , οι μεταφράσεις που παράγονται κατά το Page Walk αποθηκεύονται στο TLB και στις δευτερεύουσες Paging Structure caches. Ειδικά στην περίπτωση που για την εικονικοποίηση του Guest OS χρησιμοποιείται η τεχνική του TDP, το υλικό μπορεί να αποθηκεύσει τις μεταφράσεις των EPTs (GPA $\rightarrow$ HPA) αλλά και τις συνολικές μεταφράσεις (GVA $\rightarrow$ HPA). Με αυτόν τον τρόπο μπορούν να παραληφθούν κατά το Page Walk επίπεδα που αφορούν και τις δύο διαστάσεις, δηλαδή και επίπεδα των Page Tables του Guest αλλά και των Extended Page Tables.

Πολλαπλά Μεγέθη Σελίδων

Αν και τυπικά το υλικό μεταφράζει σελίδες σε πλαίσια με συγκεκριμένο μέγεθος, υπάρχει η υποστήριξη από το υλικό για πολλαπλά μεγέθη σελίδας. Στην αρχιτεκτονική x86-64 υπάρχει η υποστήριξη για σελίδες των 2MiB και 1GiB, που ονομάζονται Huge Pages. Όταν για ένα εύρος εικονικών διευθύνσεων έχουν χρησιμοποιηθεί Huge Pages, τότε κατά το Page Walk το υλικό καταλαβαίνει πως ένα page table entry, μέσω ενός bit που έχει τεθεί σε αυτό από το λειτουργικό, δεν δείχνει σε ένα Page Table του επόμενου επιπέδου αλλά

σε φυσική διεύθυνση μεγαλύτερου εύρους από του κανονικού. Στο Σχήμα 2.8 φαίνεται ένα τέτοιο παράδειγμα.

Σχήμα 2.8: Page walk για διεύθυνση που χρησιμοποιεί huge pages



Με την χρήση των Huge Pages μειώνεται σημαντικά η πίεση στο TLB αφού αρκούν λιγότερα entries για να μεταφραστεί ένα εύρος διευθύνσεων. Επίσης μειώνεται ο χρόνος των Page Walk αφού πλέον το υλικό χρειάζεται να διασχίσει λιγότερα επίπεδα. Τα οφέλη αυτά είναι ακόμα μεγαλύτερα σε εικονικοποιημένο περιβάλλον όταν χρησιμοποιούνται τα EPTs.

Η αρχική υποστήριξη από το Linux kernel επέτρεπε στο userspace να ζητήσει ρητά από τον πυρήνα ένα εύρος διευθύνσεων που υποστηρίζεται από Huge Pages [LWN11]. Πλέον ο πυρήνας χρησιμοποιεί διαφανώς τα Huge Pages προάγοντας ένα σύνολο από κανονικές σελίδες σε ένα Huge Page χωρίς την συνεργασία της εκάστοτε εφαρμογής. Το γνώρισμα αυτό ονομάζεται Transparent Huge Pages.

Σελιδοποίηση 5 επιπέδων

Η υπάρχουσα υλοποίηση της σελιδοποίησης στην αρχιτεκτονική x86-64 χρησιμοποιεί 4 επίπεδα από Page Tables, με μέγεθος σελίδας 4KiB. Όπως αναφέρθηκε και σε προηγούμενη ενότητα, αυτό οδηγεί σε 512 ptes ανά σελίδα που απαιτούν 9 bits από την εικονική διεύθυνση για να δεικτοδοτηθούν. Συνολικά, 36bits ορίζουν πλήρως την σελίδα στη οποία αντιστοιχεί μια διεύθυνση, και, λαμβάνοντας υπόψη και την μετατόπιση, 48bits εικονικών διευθύνσεων ορίζουν μοναδικά ένα πλαίσιο. Για να μπορέσουμε να αναφερθούμε σε παραπάνω από 256TiB που αντιστοιχούν στα 48bits, έχει υλοποιηθεί η προσθήκη ενός ακόμα επιπέδου στα Page Tables, αυξάνοντας το μέγεθος του εικονικού χώρου διευθύνσεων σε 128PiB ($= 2^{57}$B) [LWN17b].

Αυτό θα οδηγήσει σε ακόμα μεγαλύτερο κόστος κατά την εκτέλεση ενός VM με TDP αφού θα χρειάζονται 5 προσβάσεις για κάθε επίπεδο, αθροίζοντας συνολικά σε έως και 35 προσβάσεις στην μνήμη από ένα TLB miss αντί για 24 που απαιτεί η τρέχουσα υλοποίηση.

Κεφάλαιο 3

Αλλαγή Μεθόδου Σελιδοποίησης κατά τον χρόνο εκτέλεσης

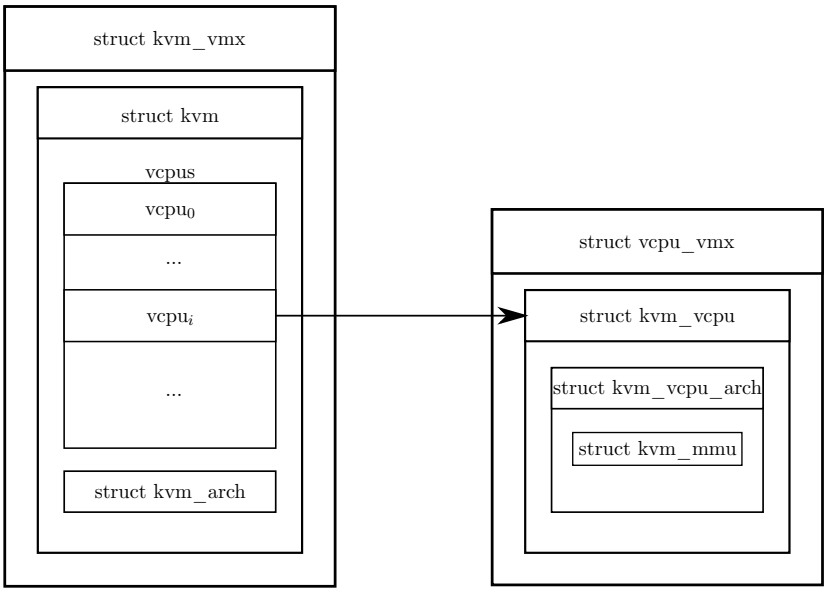
3.1 Υλοποίηση του μηχανισμού

Για την περιγραφή του μηχανισμού που υλοποιήσαμε, αναφέρουμε πρώτα τις απαραίτητες δομές που χρησιμοποιούνται από το KVM για να περιγράψει ένα VM και την MMU του, όπως και τον τρόπο με τον οποίο διαφοροποιείται το μέρος του κώδικα που εξαρτάται από την μέθοδο εικονικοποίησης της μνήμης.

3.1.1 Δομές του KVM

Κάθε VM περιγράφεται από μία δομή struct kvm, η οποία περιλαμβάνει ένα πίνακα από δομές struct kvm_vcpu που περιγράφουν μία vcpu. Οι δομές struct kvm και kvm_vcpu περιλαμβάνουν και τις αντίστοιχες δομές, struct kvm_arch και struct kvm_vcpu_arch, που περιλαμβάνουν πληροφορίες που διαφέρουν ανάλογα με την αρχιτεκτονική. Μέσα στην δομή struct kvm_vcpu υπάρχει η δομή struct kvm_mmu που περιγράφει την mmu για μία vcpu. Επίσης, οι δομές struct kvm και struct kvm_vcpu είναι ενσωματωμένες η κάθε μία σε μία άλλη δομή που περιέχει δεδομένα που διαφέρουν μεταξύ των υλοποιήσεων της Intel και της AMD ({kvm,vcpu}_vmx και {kvm,vcpu}_svm, αντίστοιχα). Στο Σχήμα 3.1 φαίνεται η ιεραρχία των δομών για την υλοποίηση της Intel (VMX).

Σχήμα 3.1: Οι δομές που χρησιμοποιούνται στο VMX για την περιγραφή ενός VM



3.1.2 Επιλογή μεθόδου κατά την εκκίνηση του VM

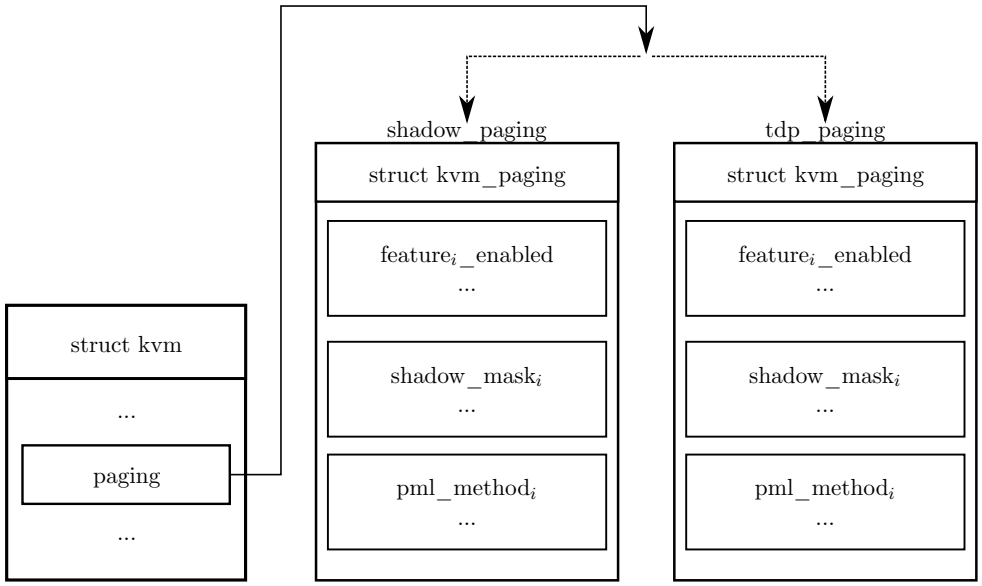
Η επιλογή της μεθόδου που θα χρησιμοποιηθεί γίνεται κατά την εισαγωγή του kvm module στον πυρήνα. Από προεπιλογή ο πυρήνας αν εντοπίσει την αντίστοιχη υποστήριξη τότε χρησιμοποιεί το TDP. Εναλλακτικά η μέθοδος μπορεί να οριστεί κατά το φόρτωμα του module στον πυρήνα.

Ως πρώτο βήμα για την υλοποίηση του μηχανισμού, μεταφέραμε όλο το global state που εξαρτάται από το paging mode, άμεσα ή μεταβατικά, σε μία δομή (struct kvm_paging, Σχήμα 3.2) και παρέχουμε στο userspace την δυνατότητα να ορίζει ποια τεχνική θα χρησιμοποιηθεί χωρίς να χρειάζεται να ξαναφορτωθεί το module με διαφορετική παράμετρο. Έτσι μπορούν να συνυπάρχουν ταυτόχρονα VMs με διαφορετική μέθοδο εικονικοποίησης των MMUs.

Η δομή αυτή περιέχει:

- Κάποιες μάσκες που χρησιμοποιούνται στα shadow page tables.
- Την πληροφορία για το αν είναι ενεργοποιημένα γνωρίσματα που είναι διαθέσιμα μόνο με χρήση του TDP, όπως EPT A/D bits, Page-Modification Logging (PML), Unrestricted Guest.
- Κάποιες μεθόδους (methods, function pointers) που αφορούν το PML, στην περίπτωση που έχει ενεργοποιηθεί.

Σχήμα 3.2: Αποσύνδεση του global state από το paging mode



3.1.3 MMU δομές

Για να υλοποιήσουμε τον μηχανισμό για την δυναμική αλλαγή μελετήσαμε τις δομές που χρησιμοποιεί το kvm για την υλοποίηση του MMU ενός VM. Όπως έχει αναφερθεί, και στις 2 περιπτώσεις τα page tables που χρησιμοποιεί το KVM ονομάζονται shadow page tables (spts) και κάθε ένα από αυτά περιγράφεται από ένα struct kvm_mmu_page. Όλα τα kvm_mmu_page βρίσκονται σε μία λίστα με όλα τα ενεργά shadow page tables (active_mmu_pages). Επίσης κάθε kvm_mmu_page βρίσκεται και σε ένα hash table ώστε να μπορεί να βρεθεί αποδοτικά ένα spt βάσει του αντίστοιχου gfn.

Η ρίζα των πιο πρόσφατα χρησιμοποιημένων shadow page tables μαζί με τον αντίστοιχο CR3 αποθηκεύονται σε μία δομή για κάθε vcpu και το αντίστοιχο kvm_mmu_page κάθε τέτοιας εγγραφής δεν επιτρέπεται να απελευθερωθεί μέχρι να αφαιρεθεί από αυτή τη δομή.

3.1.4 Υλοποίηση

Για να γίνει η δυναμική αλλαγή χρειάζεται να:

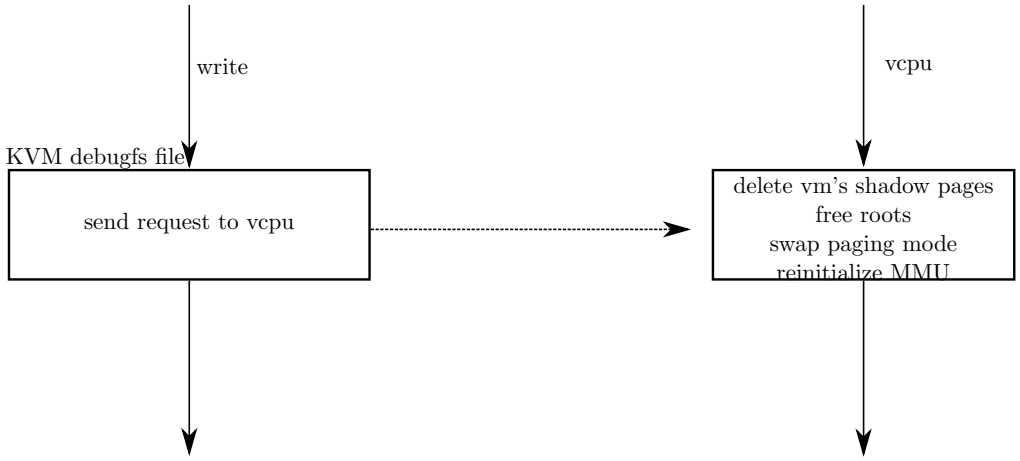
- καταστρέψουμε όλα τα προηγούμενα page tables
- αρχικοποιήσουμε ξανά όλες τις δομές που εξαρτώνται από το paging mode
- ενημερώσουμε κατάλληλα το VMCS ώστε το υλικό να χρησιμοποιήσει την νέα μέθοδο.

Για τα πρώτα δύο καταστρέφουμε όλα τα page tables και αδειάζουμε τις caches των root page tables, καταστρέφοντας και τα τελευταία page tables που υπάρχουν. Επίσης,

αλλάζουμε την kvm_paging δομή και αφήνουμε την vcpu να αρχικοποιήσει ξανά το MMU του, καλώντας τις αντίστοιχες συναρτήσεις υψηλού επιπέδου.

Για το τελευταίο, χρειάζεται να ενημερώσουμε το Exception Bitmap και το VM-execution control field του vmcs ώστε να ρυθμίσουμε αν θα προκαλούνται VM-exits κατά τα page fault, τις αναγνώσεις και εγγραφές στον cr3 και την εκτέλεση εντολών INVLPG. Επιπρόσθετα πρέπει να ενημερώσουμε τα VM-entry και VM-exit control fields του VMCS που καθορίζουν κάποιες λεπτομέρειες σχετικά με τον καταχωρητή EFER.

Σχήμα 3.3: Αλλαγή της μεθόδου εικονικοποίησης της μνήμης



Τέλος πρέπει να ανανεώσουμε το secondary execution control field του VMCS που ορίζει αν χρησιμοποιούνται τα EPTs. Όλη η διαδικασία φαίνεται στο Σχήμα 3.3.

3.1.5 Πιθανή βελτιστοποίηση

Μία βελτιστοποίηση που θα μπορούσε γίνει είναι κατά την αλλαγή από TDP σε shadow paging να μην καταστρέφονται τα spts που αφορούν το TDP αλλά να διατηρούνται παράλληλα με τα νέα, εφαρμόζοντας και σε αυτά όσες αλλαγές επιβάλουν οι MMU notifiers [LWN08] και η userspace εφαρμογή μέσων των memory slots.. Έτσι δεν θα χρειάζεται να κατασκευαστούν από την αρχή την επόμενη φορά που θα επιλεγεί το TDP ως μέθοδος σελιδοποίησης. Επιπρόσθετα τα spts που αντιστοιχούν στο TDP είναι σχετικά στατικά και έτσι δεν αναμένεται να προσθέσει σημαντικό κόστος η παράλληλη διατήρηση τους.

3.1.6 Περιορισμοί

Ένας σημαντικός περιορισμός που προέκυψε είναι η αδυναμία χρήσης παραπάνω από μίας vcpu σε ένα VM που θα χρησιμοποιήσει τον μηχανισμός. Συγκεκριμένα, στην περίπτωση πολλαπλών vcpus, κατά την διάρκεια της αλλαγής υπήρχαν περιπτώσεις στις οποίες προέκυπταν προβλήματα στον guest πυρήνα. Το πρόβλημα αυτό οφείλεται σε bug της υλοποίησης μας που δεν καταφέραμε να εντοπίσουμε. Χρησιμοποιώντας μόνο μία vcpu το πρόβλημα δεν εμφανίζεται, και για αυτό χρησιμοποιήσαμε τον μηχανισμό μόνο για VMs με μία vcpu.

3.2 Πολιτική

Στην υλοποίηση μας προσφέρουμε τη δυνατότητα αλλαγής του paging mode ενός VM κατά τον χρόνο εκτέλεσης μέσω ενός ειδικού αρχείου στο kvm debugfs. Για να την αξιοποιήσουμε, υλοποιήσαμε μια εφαρμογή στο χώρο χρήστη που συλλέγει, χρησιμοποιώντας το εργαλείο perf, κάποιες μετρικές για το VM που μας ενδιαφέρει ανά κάποιο σταθερό χρονικό διάστημα.

Επιλέγαμε να εξετάσουμε και να αξιολογήσουμε την επίδοση του μηχανισμό μας βάσει δύο διαφορετικών πολιτικών. Η πρώτη πολιτική συγκρίνει το κόστος που εισάγει στην εκτέλεση της εφαρμογή με κάποια στατικά κατώφλια και με βάση τη σύγκριση αυτή μεταβάλει μία τιμή που αντικατοπτρίζει τον βαθμό εκτίμησης για το ποια μέθοδος θα ήταν η βέλτιστη. Η αλλαγή γίνεται όταν αυτή η τιμή πάρει την μέγιστη ή την ελάχιστη τιμή της, ανάλογα με την τρέχουσα μέθοδο σελιδοποίησης. Η δεύτερη πολιτική αλλάζει δυναμικά τα κατώφλια που χρησιμοποιούνται στη σύγκριση, επιβραβεύοντας ή τιμωρώντας αλλαγές μεθόδου που οδήγησαν στην αύξηση του συνολικού κόστους της σελιδοποίησης.

Μετρικές

Οι μετρικές που χρησιμοποιούμε για την απόφαση αλλαγής μεθόδου είναι:

- Το ποσοστό των κύκλων που η vcpu βρίσκεται στον host πυρήνα, που αποτελεί μια προσέγγιση του χρόνου που αφιερώνει ο VMM στην διαχείριση του MMU : $C_{VMM\%}$
- Το ποσοστό των κύκλων που οι vcpus αφιερώνουν στα page walks : $C_{PW\%}$
- Το IPC (Instructions per Cycle) που μας δίνει μία εκτίμηση για το αν η αλλαγή που έγινε βελτίωσε της επίδοση του VM ώστε να αναπροσαρμόσουμε κατάλληλα τα κατώφλια.

Overhead και αποφυγή ταλαντώσεων

Η αλλαγή του paging mode προσθέτει ένα επιπλέον κόστος στην εκτέλεση του VM επειδή πέρα από την αποδέσμευση των page tables και την δημιουργία νέων, το VM θα υποστεί κάποια VM-exits, αφού τα shadow page tables θα είναι άδεια. Για να αποφύγουμε τις ταλαντώσεις που θα προκαλούσε, προσθέτουμε ένα μικρό διάστημα μετά από κάθε αλλαγή κατά το οποίο ο μηχανισμός δεν λαμβάνει αποφάσεις.

3.2.1 Πολιτική με Σταθερά κατώφλια

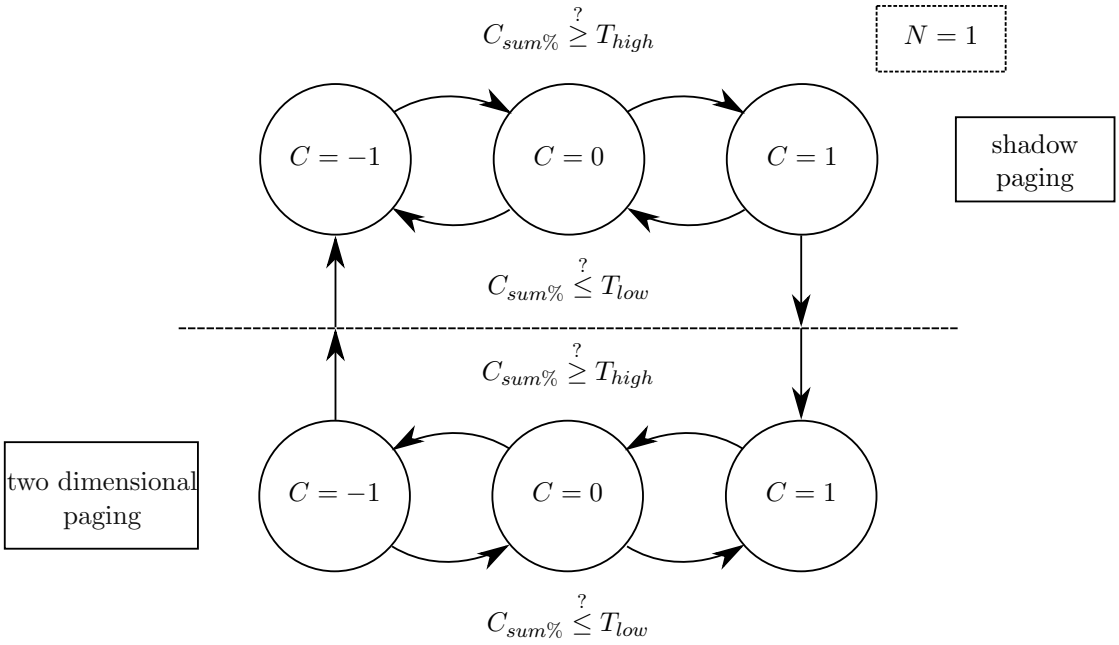
Η πρώτη πολική που εφαρμόσαμε αλλάζει το paging mode με βάση δύο κατώφλια — T_{high} και T_{low} — και ένα βοηθητικό μετρητή C που έχει εύρος από $-N$ έως N, και λειτουργεί ως δείκτης βεβαιότητας για την αλλαγή.

Συγκεκριμένα, όταν χρησιμοποιούμε το shadow paging (αντίστοιχα TDP) και το $C_{sum\%} = C_{VMM\%} + C_{PW\%}$ ξεπεράσει την τιμή T_{high} τότε αυξάνουμε (μειώνουμε) την τιμή του C,

ενώ όταν είναι μικρότερο από το κάτω όριο T_{low} τότε μειώνουμε (αυξάνουμε) την τιμή του C. Η αλλαγή από shadow σε TDP γίνεται όταν ο μετρητής φτάσει στη μέγιστη τιμή N, ενώ η αλλαγή από TDP σε shadow γίνεται όταν φτάσει στην ελάχιστη τιμή $-N$.

Αμέσως μετά την αλλαγή ο μηχανισμός σταματάει να μεταβάλει την τιμή του C για $2 \times Ts$ ώστε να μην παρερμηνευτεί η αναμενόμενη αύξηση του overhead αμέσως μετά την αλλαγή ως αποτέλεσμα της νέας μεθόδου σελιδοποίησης και προκληθεί μια συνεχής εναλλαγή μεταξύ των δύο μεθόδων. Η πολιτική έχει ως παράμετρο τη μέγιστη τιμή N του μετρητή και για $N = 1$ συνοψίζεται στο Σχήμα 3.4

Σχήμα 3.4: Πολιτική με σταθερά κατώφλια



3.2.2 Πολιτική με Δυναμικά κατώφλια

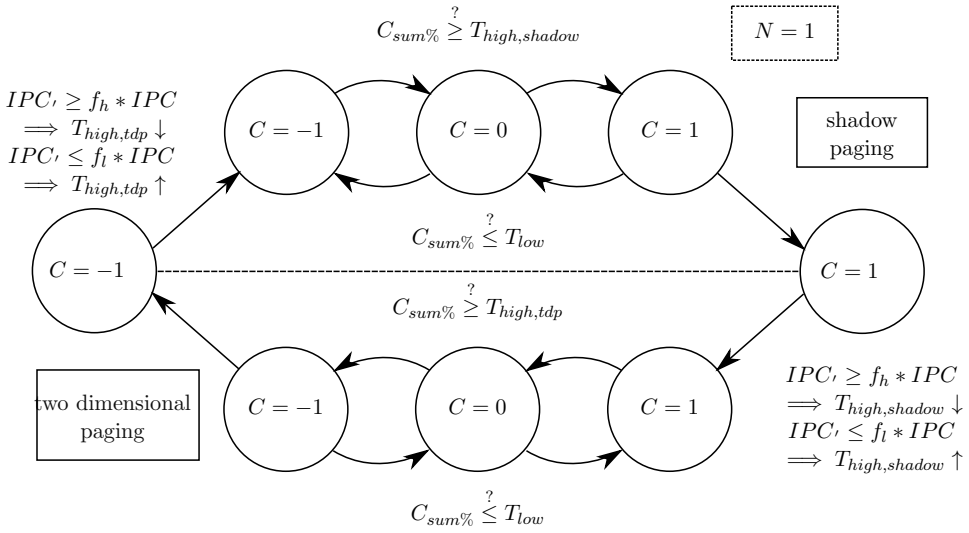
Η δεύτερη πολιτική ακολουθεί την ίδια ιδέα με την πρώτη αλλά επιπλέον προσπαθεί να προσαρμόζει δυναμικά τα κατώφλια με βάση την παρατηρούμενη αλλαγή στο συνολικό κόστος, παρόμοια με την προσέγγιση των Bae κ.ά. [Bae11].

Πλέον υπάρχουν δύο κατώφλια, $T_{high,shadow}$ και $T_{high,tdp}$ που ξεκινούν με την ίδια τιμή και όταν αμέσως μετά από μία αλλαγή παρατηρείται αύξηση ή μείωση της επίδοσης του VM, όπως φαίνεται από την τιμή του IPC, τότε αυξάνεται ή μειώνεται το αντίστοιχο κατώφλι που οδήγησε στην αλλαγή.

Για παράδειγμα, αν το κατώφλι $T_{high,tdp}$ έχει την τιμή 10, και μετά την αλλαγή από TDP σε shadow παρατηρηθεί αύξηση του IPC, τότε το κατώφλι $T_{high,tdp}$ μειώνεται και παίρνει π.χ. την τιμή 8.5 ώστε την επόμενη φορά η αλλαγή σε shadow να γίνει πιο σύντομα. Το αντίστροφο συμβαίνει στην περίπτωση που η τιμή του IPC μειωθεί.

Η μέτρηση της νέα τιμής του IPC γίνεται αφού περάσουν Ts ώστε να μην επηρεαστεί σημαντικά από την αρχική αναμενόμενη μείωσή του αμέσως μετά την αλλαγή. Η πολιτική πάλι έχει ως παραμέτρους το N αλλά και τα f_h και f_l που καθορίζουν το πόσο εύκολα μεταβάλλονται τα κατώφλια. Το μέγεθος της μεταβολής των κατωφλιών εξαρτάται από τον λόγο $\frac{IPC'}{IPC}$. Στο Σχήμα 3.5 φαίνεται η λειτουργία της πολιτικής για $N = 1$.

Σχήμα 3.5: Πολιτική με δυναμικά κατώφλια



Κεφάλαιο 4

Αξιολόγηση

4.1 Μεθοδολογία

Για την αξιολόγηση των δύο μεθόδων εκτελέσαμε κάποια benchmarks από τις σουίτες μετροπρογραμμάτων Spec2017 και PARSEC. Χρησιμοποιήσαμε το QEMU (έκδοση 4.1.0), τροποποιώντας το ελάχιστα ώστε να επιλέγεται η μέθοδος σελιδοποίησης κατά την εκκίνηση του VM, για να εκτελέσουμε ένα VM με 13GiB μνήμη και μία cpu. Τα στοιχεία του μηχανήματος που είχαμε στη διάθεσή μας φαίνονται στον Πίνακα 4.1. Για την συλλογή των στατιστικών χρησιμοποιήσαμε το εργαλείο perf (έκδοση 4.19.67 στον host και έκδοση 4.19.98 στον guest), εκτελώντας το και μέσα στο VM για την εκάστοτε εφαρμογή αλλά ταυτόχρονα και στο host μηχάνημα. Και τα δύο μηχανήματα διέθεταν τον πυρήνα Linux (έκδοση 4.19.67).

Πίνακας 4.1: Χαρακτηριστικά του μηχανήματος που χρησιμοποιήθηκε για τις μετρήσεις

Επεξεργαστής	Intel Core i5-7600, 1 thread/core, 4 cores, 3.50GHz
Μνήμη	16 GiB DDR4 2400 MHz
L1 DTLB, 2M/4M pages	4-way, 32 entries
L1 DTLB, 1G pages	4-way, 4 entries
L1 DTLB, 4K pages	4-way, 64 entries
L1 ITLB, 2M/4M pages	fully associative, 8 entries
L1 ITLB, 4K pages	8-way, 128 entries
L2 TLB 4K/2M pages	6-way, 1536 entries

Για τον υπολογισμό του κόστους που εισάγει η εικονικοποίηση της μνήμης, συγκρίνουμε τις εκτελέσεις των benchmark σε εικονικοποιημένο μηχάνημα με την εκτέλεση του ίδιου benchmark στο φυσικό μηχάνημα στο οποίο έχουμε ενεργοποιήσει τα transparent huge pages, ώστε να προσεγγίσουμε μια ιδανική εκτέλεση στην οποία ο χρόνος που περνάει η εφαρμογή στα Page Walks να έχει σχεδόν μηδενιστεί.

Συγκεκριμένα υπολογίζουμε τους ιδανικούς κύκλους εκτέλεσης μίας εφαρμογής ως $C_{Ideal} = C^{2M}_{Total} - C^{2M}_{PW}$. Στον Πίνακα 4.2 φαίνεται πως έχουμε πάρει τις μετρικές που χρειαζόμαστε από το perf ενώ στον Πίνακα 4.3 δίνουμε τον τρόπο που υπολογίζουμε το

κόστος ως προς την ιδανική εκτέλεση. Η έκδοση του πυρήνα που χρησιμοποιήσαμε στο host και στο guest μηχάνημα είναι η 4.19.67.

Πίνακας 4.2: Μετρικές μαζί με το αντίστοιχο perf event και το που εκτελέστηκε το perf.

Μετρική	**Perf event**	**Μηχάνημα**
$Cycles_{PW}^{Load}$	$cpu/event = 0x08, umask = 0x10, cmask = 0x01/$	Guest
$Cycles_{PW}^{Store}$	$cpu/event = 0x49, umask = 0x10, cmask = 0x01/$	Guest
$Cycles$	$r003c$	Guest
$Cycles_H$	$r003c : H$	Host
$Cycles_G$	$r003c : G$	Host

Πίνακας 4.3: Υπολογισμός του κόστος της εικονικοποίησης ως προς την native εκτέλεση

Overhead	**Υπολογισμός**
$Overhead_{VMM}(\%)$	$100 \times \frac{Cycles_H}{Cycles_{Ideal}}$
$Overhead_{PW}(\%)$	$100 \times \frac{Cycles - Cycles_{Ideal}}{Cycles_{Ideal}}$

Αξίζει να σημειωθεί πως στην περίπτωση του TDP, ένα VM το οποίο έχει μόλις ξεκινήσει και δεν έχει χρησιμοποιήσει σημαντικό μέρος της μνήμης του θα υποστεί μεγαλύτερη επιβάρυνση από ένα VM που έχει προσπελάσει όλη τη φυσική του μνήμη. Αυτό συμβαίνει επειδή τα Extended Page Tables κατασκευάζονται με οκνηρό τρόπο κατά τις πρώτες προσβάσεις στην φυσική μνήμη του Guest, μετά τις οποίες οι προσβάσεις στη μνήμη παύουν να προκαλούν VM exits.

Για να λάβουμε υπόψη αυτό το γεγονός, εκτελούμε δύο φορές τα benchmarks στην περίπτωση του TDP. Στην μία εκτέλεση ξεκινάμε ένα VM και αμέσως εκτελούμε το benchmark (VM μικρής διάρκειας, short lived VM), ενώ στη δεύτερη ξεκινάμε ένα VM και εκτελούμε ένα microbenchmark που προσπελαύνει όλη τη μνήμη του VM πριν εκτελέσουμε το κανονικό benchmark, προσομοιώνοντας με αυτό τον τρόπο ένα VM που εκτελείται για ώρα και τα Extended Page Tables του Host έχουν δημιουργηθεί πλήρως (long lived VM).

Η παρουσίαση και της περίπτωσης του VM μεγάλης διάρκειας γίνεται μόνο στα αρχικά αποτελέσματα που αφορούν την αξιολόγηση των δύο αρχικών μεθόδων. Στα υπόλοιπα αποτελέσματα παρουσιάζουμε μόνο την περίπτωση του VM μικρής διάρκειας για όλες τις μεθόδους που εξετάζουμε.

4.1.1 Hugepages

Στο σύνολο των πειραμάτων που κάναμε, δεν έχουμε χρησιμοποιήσει τα transparent huge pages επειδή με το μηχάνημα που διαθέτουμε και τα benchmarks τα οποία εξετάζουμε δεν θα βλέπαμε σημαντικό κόστος σε ότι αφορά τα Page Walks. Παρ' όλα αυτά έχει παρατηρηθεί πως ακόμα και με μεγαλύτερες σελίδες το κόστος σε διάφορους φόρτους εργασίας δεν είναι αμελητέο [Gand14, Pham15, Gand16]

4.2 Αποτελέσματα

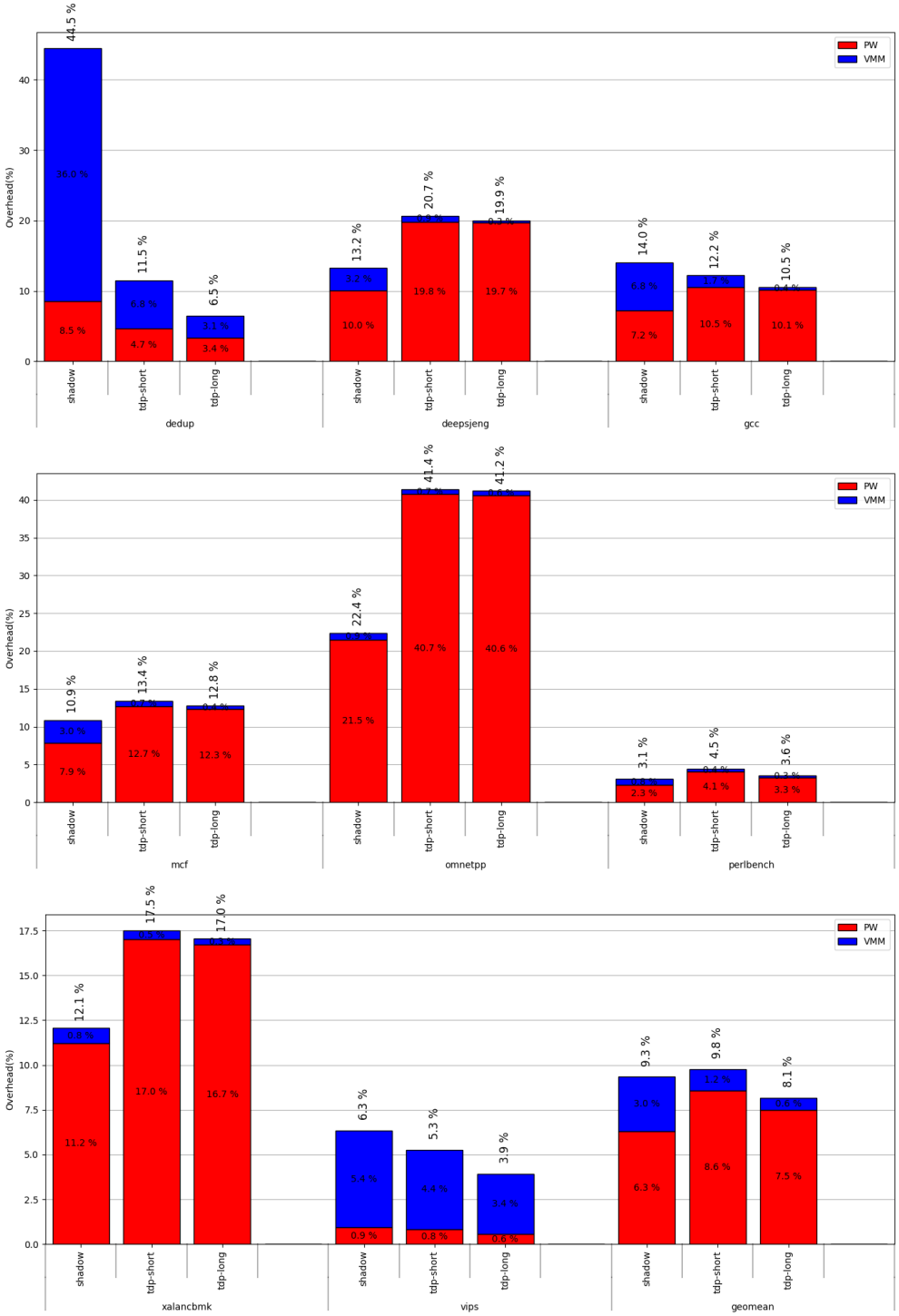
4.2.1 Αρχική σύγκριση των δύο μεθόδων

Αρχικά, συγκρίνουμε το κόστος που εισάγουν οι δύο μέθοδοι, χωρίζοντάς το σε κόστος που οφείλεται στην διαχείριση του VM από τον VMM, και στο κόστος που οφείλεται στα Page Walks (PW). Παρουσιάζουμε τα αποτελέσματά μας στο Σχήμα 4.1. Όπως και στα επόμενα σχήματα, έχουμε προσθέσει τον γεωμετρικό όρο των μετρήσεων για όλα τα benchmarks που εκτελέσαμε (geomean).

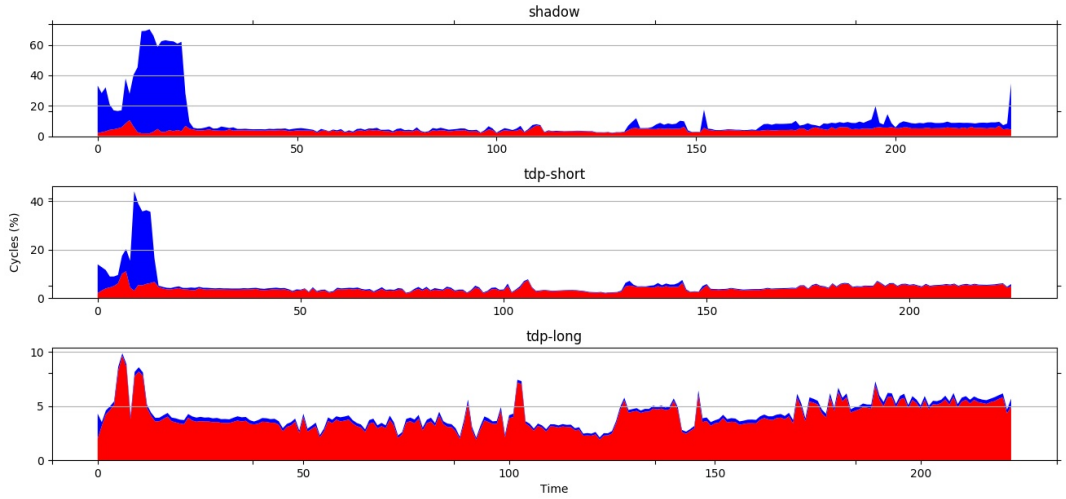
Παρατηρούμε πως και με τις δύο μεθόδους το κόστος κυμαίνεται από μικρό (4%) έως και αρκετά σημαντικό (πάνω από 40%). Σχεδόν σε όλες τις περιπτώσεις το VM μεγάλης διάρκειας υποφέρει λιγότερο, κυρίως λόγω του μειωμένου κόστους από την διαχείριση του VM από τον VMM.

Για ένα συγκεκριμένο benchmark, το gcc, παρουσιάζουμε στο Σχήμα 4.2 το ποσοστό των κύκλων της εφαρμογής που αντιστοιχεί σε παρεμβάσεις από το VMM και σε Page Walks στη διάρκεια του χρόνου.

Πέρα από τις διαφορές μεταξύ των μεθόδων, βλέπουμε πως το ποσοστό του χρόνου που αφιερώνεται στον VMM και σε Page Walks μεταβάλλεται κατά την διάρκεια της εκτέλεσης.



Σχήμα 4.1: Κόστος που εισάγει η εικονικοποίηση σε σχέση με την native εκτέλεση.



Σχήμα 4.2: Ποσοστό των κύκλων που το gcc περνάει σε Page Walks (κόκκινο) και στον VMM (μπλε).

4.2.2 Σταθερά Κατώφλια

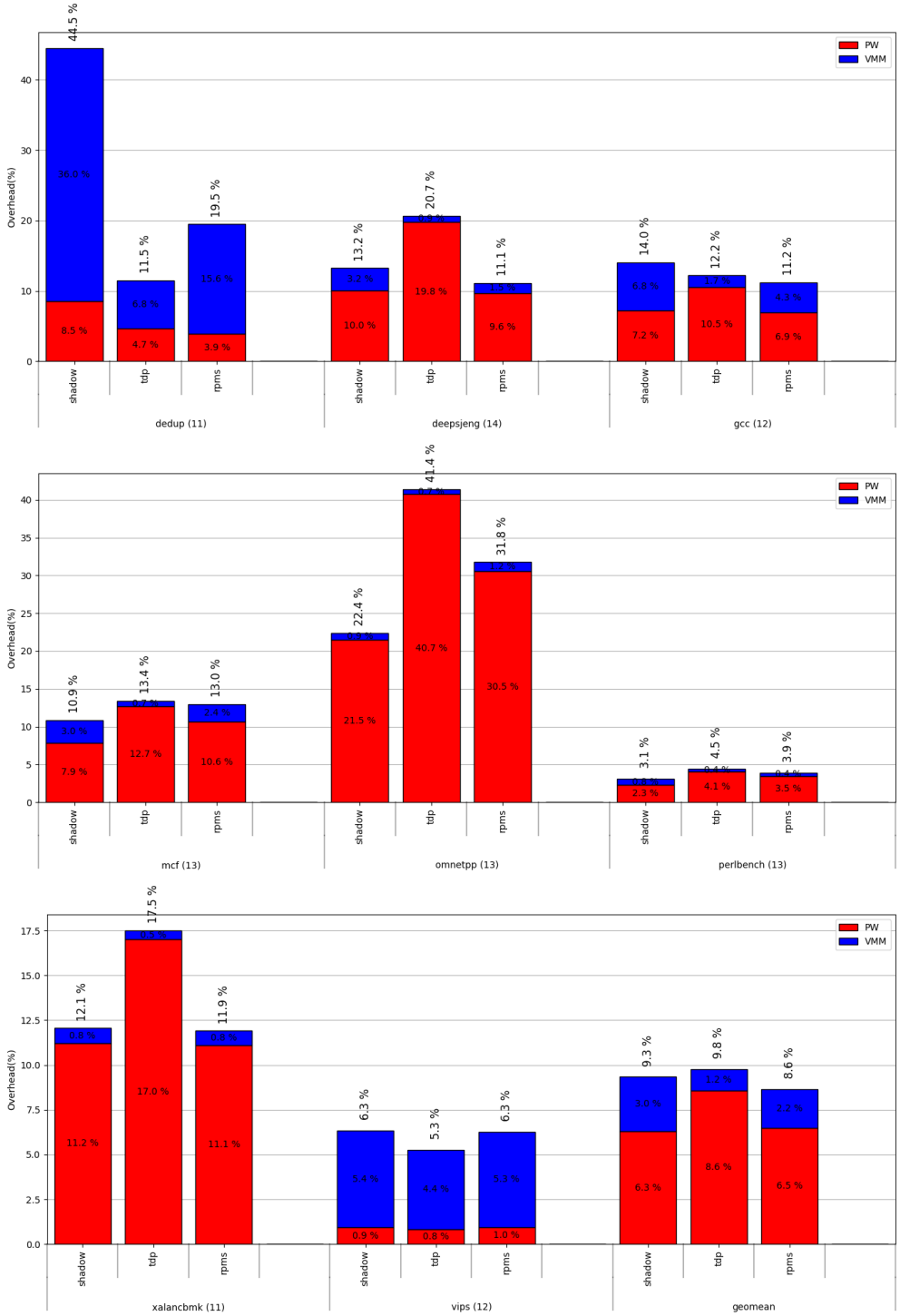
Αρχικά, εκτελέσαμε τα benchmarks για διάφορες στατικές τιμές των πάνω κατωφλιών (T_{high}) και για $N = 4$, $T_{low} = 3$, $T = 1s$. Το χρονικό διάστημα T μεταξύ των μετρήσεων το επιλέξαμε ώστε να μην είναι πολύ μεγάλο ώστε να μπορεί να παρακολουθήσει τις αλλαγές στον φόρτο εργασίας του guest αλλά ούτε πολύ μικρό ώστε να μην προκαλεί την αλλαγή μία παροδική αιχμή της αντίστοιχης μετρικής.

Στο Σχήμα 4.3 παρουσιάζουμε το κόστος εικονικοποίησης των benchmarks, με ενεργοποιημένο τον μηχανισμό μας, έχοντας επιλέξει το βέλτιστο T_{high} από το εύρος τιμών 3 έως 20 που δοκιμάσαμε. Ως αρχική μέθοδος σελιδοποίησης για κάθε benchmark επιλέχτηκε αυτή που παρουσιάζει το μεγαλύτερο overhead, και είναι το shadow paging για τα gcc, dedup και vips, και το tdp για τα υπόλοιπα.

Την εκτέλεση με τον μηχανισμό μας, και με χρήση της αντίστοιχης πολιτικής, την αναφέρουμε ως rpms-shadow ή rpms-tdp (Runtime-aware Paging Mode Switching), ανάλογα με την αρχική μέθοδο με την οποία ξεκινάμε.

Παρατηρούμε πως πέρα από διαφορετικά βέλτιστα κατώφλια που χρειάζονται για κάθε εφαρμογή, η πολιτική αυτή έχει το μειονέκτημα ότι θέτει και τα δύο κατώφλια σε ίση αρχική τιμή, κάτι που δεν είναι ιδανικό όπως φαίνεται στην περίπτωση του omnetpp, μιας και ενώ η βέλτιστη μέθοδος θα ήταν η στατική επιλογή του shadow paging, η πολιτική οδηγεί σε πολλές εναλλαγές επειδή δεν λαμβάνει υπόψη την πραγματική βελτίωση της επίδοσης.

Η πολιτική που προτείνουμε παρακάτω διορθώνει το πρόβλημα αυτό, αλλάζοντας δυναμικά τα κατώφλια με βάση την παρατηρούμενη αλλαγή στο IPC αμέσως μετά την αλλαγή.



Σχήμα 4.3: Κόστος που εισάγει η εικονικοποίηση σε σχέση με την native εκτέλεση, με ενεργοποιημένο τον μηχανισμό μας με πολιτική σταθερών κατωφλιών. Σε παρένθεση έχουμε σημειώσει το κατώφλι T_{high} που χρησιμοποιήσαμε για κάθε benchmark.

4.2.3 Δυναμικά Κατώφλια

Αφού παρατηρήσαμε πως διαφορετικές εφαρμογές θα έχουν άλλες βέλτιστες τιμές κατωφλιών, επαναλαμβάνουμε τις μετρήσεις μας χρησιμοποιώντας την δεύτερη πολιτική με την οποία τα κατώφλια προσαρμόζονται δυναμικά. Ως αρχική τιμή πάνω κατωφλίου θέτουμε $T_{high} = 12$, για τις ευαισθησίες αλλαγής κατωφλιών επιλέγουμε $f_h = 1.1, f_l = 0.9$ και οι υπόλοιπες παράμετροι διατηρούν τις ίδιες τιμές ($N = 4$, $T_{low} = 3$, $T = 1s$).

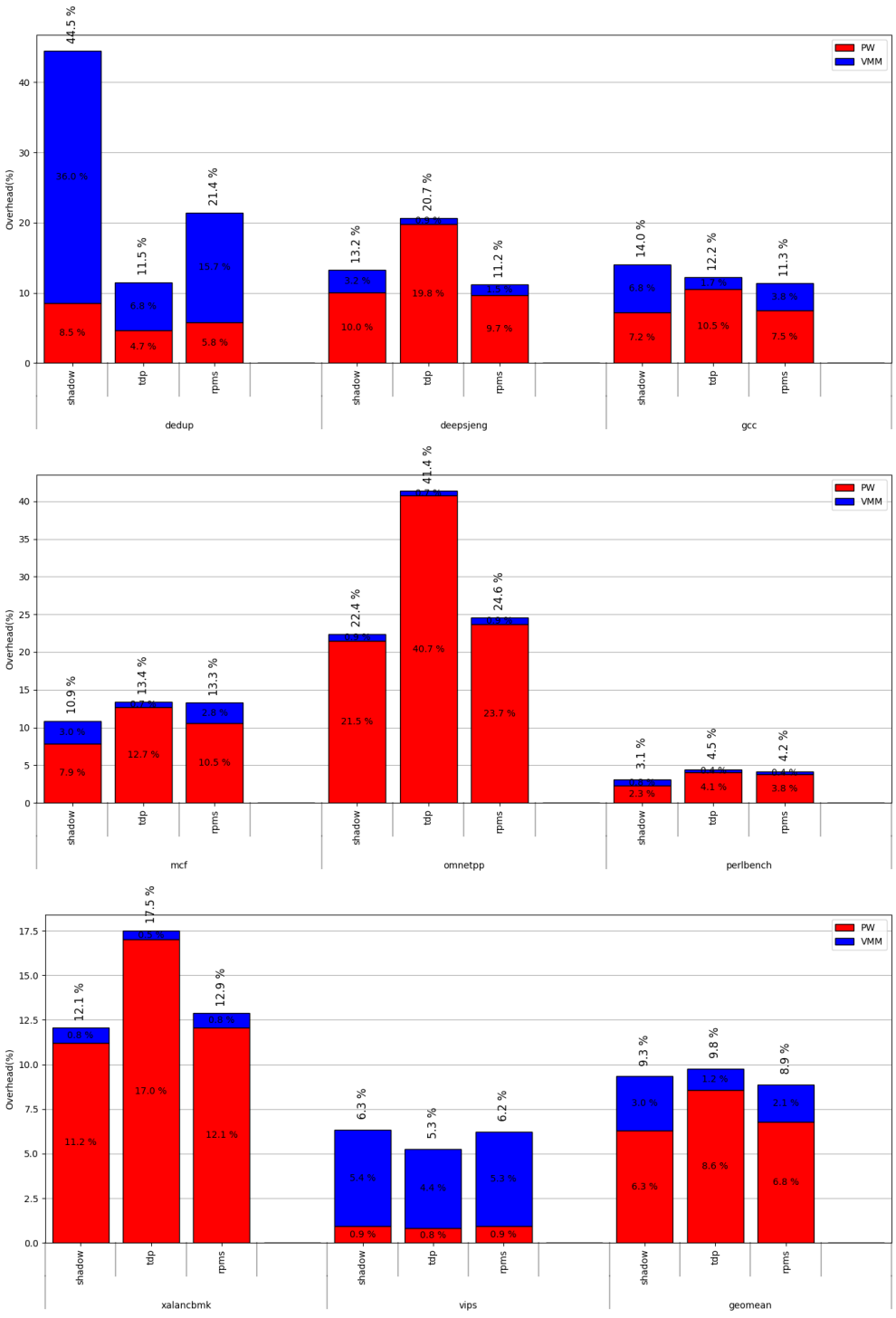
Στο Σχήμα 4.4 συγκρίνουμε το κόστος που εισάγει η εικονικοποίηση όταν έχουμε ενεργοποιήσει τον μηχανισμό μας με την πολιτική που αναφέραμε, με το κόστος των υπόλοιπων δύο μεθόδων, χωρίζοντάς το πάλι σε αυτό που οφείλεται σε Page Walks και σε αυτό που αντιστοιχεί σε παρεμβάσεις του VMM.

Παρατηρούμε πως η πολιτική μας καταφέρνει σε γενικές γραμμές να επιτύχει επίδοση κοντά στην στατικά βέλτιστη μέθοδο σελιδοποίησης, ή και καλύτερη από αυτή σε μερικές περιπτώσεις. Εξαίρεση αποτελεί το mcf, όπου παρατηρούνται κάποιες μεγάλες αυξήσεις στο συνολικό κόστος της εικονικοποίησης και στις δύο μεθόδους, και έτσι προκαλείται αλλαγή της μεθόδου, που όμως δεν καταφέρνει να βελτιώσει την επίδοση.

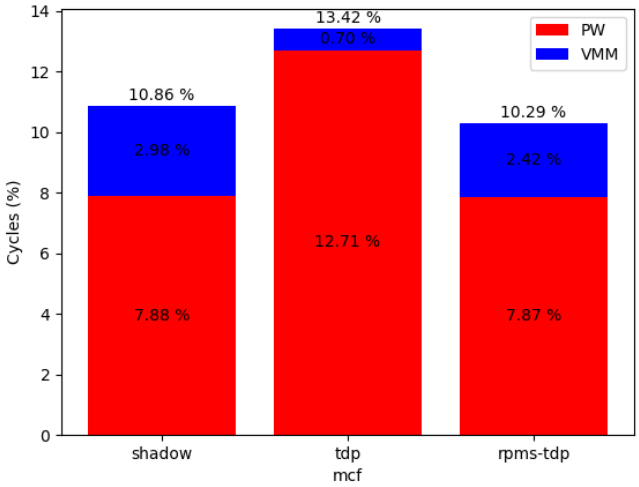
Ένας τρόπος για να αποφύγουμε αυτό το φαινόμενο είναι να αλλάζει η τιμή του μετρητή C μόνο με βάση μία από τις δύο μετρικές για κάθε μέθοδο ($C_{VMM\%}$ για το shadow paging, $C_{PW\%}$ για το tdp). Πράγματι, έτσι καταφέρνουμε να πετύχουμε επίδοση καλύτερη και από αυτή της βέλτιστης στατικής τεχνικής (shadow paging), όπως φαίνεται στο Σχήμα 4.5.

Επιπλέον, στα vips και perlbench, επειδή το συνολικό κόστος είναι σχετικά μικρό και στο shadow και στο TDP, ο μηχανισμός δεν ενεργοποιείται ποτέ και έτσι δεν υπάρχει βελτίωση σε σχέση με την αρχική μέθοδο. Στο dedup αν και γίνεται γρήγορα αντιληπτή η ανάγκη για αλλαγή της μεθόδου σελιδοποίησης, ο χρόνος εκτέλεσης του benchmark είναι πολύ μικρός για να καταφέρει να γίνει απόσβεση του κόστους της αλλαγής και της αρχικής εκτέλεσης με την κοστοβόρα μέθοδο.

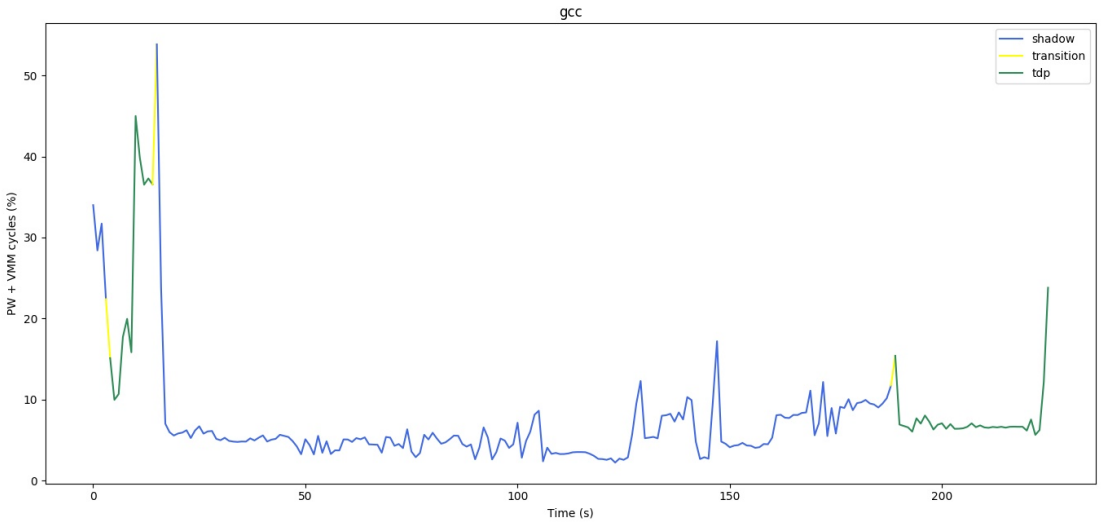
Για το gcc βλέπουμε πάλι την ανάλυση των κύκλων σε VMM και Page Walks, με ενεργοποιημένο αυτή τη φορά τον μηχανισμός μας (Σχήμα 4.6). Σε δύο σημεία της εκτέλεσής του επιλέγεται να γίνει αλλαγή της μεθόδου σελιδοποίησης, την πρώτη φορά εξαιτίας του μεγάλου χρόνου που αφιερώνεται στον VMM και την δεύτερη φορά λόγω του αυξημένου κόστους των Page Walks.



Σχήμα 4.4: Κόστος που εισάγει η εικονικοποίηση σε σχέση με την native εκτέλεση, με ενεργοποιημένο τον μηχανισμό μας με πολιτική δυναμικών κατωφλιών.



Σχήμα 4.5: Επίδοση του mcf με πολιτική που κάνει αλλαγές με βάση μία μόνο μετρική κάθε φορά.



Σχήμα 4.6: Συνολικό ποσοστό των κύκλων που περνάει το gcc στον VMM και σε Page Walks, μαζί με τις αλλαγές μεθόδου σελιδοποίησης που προκαλούνται.

Κεφάλαιο 5

Σχετική Δουλειά

Σε αυτό το κεφάλαιο παρουσιάζουμε τις δουλειές που έχουν γίνει σχετικά με την μελέτη της επίδοσης των 2 μεθόδων εικονικοποίησης της μνήμης και τον συνδυασμό τους μέσω διάφορων πολιτικών, και αναφέρουμε άλλες προσεγγίσεις που έχουν εφαρμοστεί για την μείωση του κόστους της εικονικοποίησης της μνήμης ή και γενικότερα του κόστους που της εικονικής μνήμης και σε native περιβάλλον.

5.1 Εναλλαγή μεθόδων εικονικοποίησης της μνήμης

Η ιδέα για τον συνδυασμό των 2 μεθόδων εικονικοποίησης της μνήμης δεν είναι νέα και έχει ήδη υλοποιηθεί σε διάφορους hypervisors. Οι Wang κ.ά. παρατήρησαν ότι ούτε το shadow paging ούτε το TDP είναι πάντα η καλύτερη επιλογή και υπάρχουν περιπτώσεις που η μία τεχνική έχει σαφές πλεονέκτημα έναντι της άλλης [Wang11]. Στην δουλειά τους έχουν υλοποιήσει ένα μηχανισμό για τον Xen hypervisor ώστε να μπορεί δυναμικά να επιλέγεται η τεχνική που αναμένεται να υπερτερεί για το αντίστοιχο workload. Η απόφαση λαμβάνεται με βάση την σύγκριση 2 μετρικών, της συχνότητας των page faults του guest και των TLB misses. Όταν παρατηρείται υψηλή συχνότητα από page faults τότε γίνεται η μετάβαση σε TDP, ενώ στην περίπτωση συχνών TLB misses γίνεται η αντίστροφη μετάβαση σε shadow paging. Όταν και οι δύο μετρικές είναι υψηλές χρησιμοποιείται ο λόγος τους για να αποφασιστεί αν πρέπει να γίνει αλλαγή στην μέθοδο εικονικοποίησης. Οι τιμές με τις οποίες συγκρίνονται οι μετρικές είναι προκαθορισμένες και έχουν τεθεί με χρήση μεθόδων Μηχανικής Μάθησης.

Επόμενες δουλειές βελτιώνουν την προηγούμενη, αντικαθιστώντας τους ευριστικούς κανόνες με μηχανισμούς που χρησιμοποιούν Μηχανές Διανυσμάτων Υποστήριξης [Kuan14], ή εφαρμόζοντας το πρόβλημα της δυναμική επιλογής μεθόδου στο μοντέλο του Contextual Bandit, που δεν απαιτεί προηγούμενη ανάλυση του φόρτου που θα τρέξει [Hieb18].

Η δυναμική αλλαγή μεταξύ του shadow paging και του TDP έχει υλοποιηθεί και για το KVM [Zhan17]. Η απόφαση για την μετάβαση από shadow paging σε TDP βασίζεται στον πλήθος των page faults και των VM exits, ενώ η μετάβαση από TDP σε shadow paging εξαρτάται από τον ρυθμό των TLB misses σε συνδυασμό με τις προηγούμενες δύο μετρικές ώστε να αποφευχθούν αλλαγές που δεν θα επέφεραν κέρδος σε ότι αφορά την επίδοση. Οι N τελευταίες τιμές των μετρικών συγκρίνονται με μία προκαθορισμένη τιμή και η αλλαγή γίνεται μόνο αν ένα μεγάλο ποσοστό των συγκρίσεων έχει την αντίστοιχη τιμή.

Τέλος, ο μηχανισμός της δυναμικής αλλαγής έχει υλοποιηθεί για το Palacios VMM από τους Bae κ.ά. [Bae11]. Πάλι η πολιτική βασίζεται στην μέτρηση του ρυθμού των TLB misses και στα VM exits που σχετίζονται με την μνήμη. Σε αυτή την περίπτωση όμως τα όρια των τιμών για τις 2 μετρικές καθορίζονται δυναμικά. Συγκεκριμένα, μετά από κάθε αλλαγή υπάρχει μία μεταβατική κατάσταση κατά την οποία αν παρατηρηθεί αύξηση της επίδοσης, όπως παρατηρείται από το CPI, τότε μειώνεται το αντίστοιχο κατώφλι, ενώ σε αντίθετη περίπτωση αυξάνεται ώστε η αλλαγή να γίνει μόνο αν η μετρική υπερβεί μια μεγαλύτερη τιμή από πριν.

5.2 Διαφορετικές προσεγγίσεις

Μία άλλη προσέγγιση, το Agile Paging, έχει ακολουθηθεί από τους Gahndhi κ.ά., οι οποίοι αξιοποιούν το γεγονός ότι τα πρώτα επίπεδα των page tables είναι στατικά και αλλάζουν πιο σπάνια σε σχέση με τα χαμηλότερα επίπεδα [Gand16]. Στο Agile Paging, οι δομές που χρησιμοποιούν οι 2 τεχνικές συνυπάρχουν, και κατά το page fault το page walk ξεκινάει όπως στην περίπτωση του shadow paging. Σε οποιοδήποτε επίπεδο όμως το λειτουργικό μπορεί να υποδείξει στο υλικό να συνεχίσει το page walk με την χρήση των EPTs. Έτσι και οι 2 τεχνικές αποτελούν ειδικές περιπτώσεις του Agile Paging. Με αυτό τον τρόπο μόνο εγγραφές στα υψηλότερα επίπεδα των page tables προκαλούν vm exits, με τα επίπεδα αυτά να μην χρειάζονται τα EPTs για να προσπελαστούν. Το επίπεδο στο οποίο γίνεται η αλλαγή καθορίζεται δυναμικά από μία πολιτική που λαμβάνει υπόψη το πλήθος των εγγραφών και την συχνότητα των προσβάσεων που γίνονται στα page tables.

Άλλες δουλειές προτείνουν πιο επεμβατικές παρεμβάσεις, αλλάζοντας την δομή των Page Tables ή του τρόπου που γίνεται το Page Walk. Συγκεκριμένα, έχει προταθεί η αλλαγή της δομής των EPTs και έχει ερευνηθεί η χρήση πινάκων κατακερματισμού (hash tables) αντί για δενδρική δομή [Hoan10, Yani16], όπως και η επιτάχυνση της μετάφρασης διευθύνσεων μέσω direct segments [Gand14] ή δευτερευόντων απεικονίσεων [Kara15] που συνυπάρχουν με τις υπάρχουσες δομές και δεν απαιτούν την συνεργασία των εφαρμογών. Τέλος, οι Margaritov κ.ά. προτείνουν την εφαρμογή τεχνικών μηχανικής μάθησης για την εκμάθηση των απεικονίσεων ώστε να προβλεφθεί η ζητούμενη διεύθυνση και να αποφευχθούν τα κοστοβόρα Page Walks [Marg18].

Οι προτάσεις αυτές έχουν το μειονέκτημα ότι απαιτούν από μικρές έως και αρκετά σημαντικές αλλαγές στο υπάρχον υλικό.

Κεφάλαιο 6

Συμπεράσματα και μελλοντικές επεκτάσεις

Και οι δύο υπάρχουσες τεχνικές για την εικονικοποίηση της μνήμης εισάγουν ένα σημαντικό κόστος στην εκτέλεση μίας εικονικής μηχανής. Σε αυτή την εργασία μελετήσαμε τις δύο τεχνικές, υλοποιήσαμε ένα μηχανισμό ώστε να μπορεί να αλλάξει η τεχνική που χρησιμοποιείται κατά τον χρόνο εκτέλεσης και προτείναμε δύο πολιτικές που τον οδηγούν ώστε να βελτιωθεί η επίδοση του εικονικού μηχανήματος.

Πιθανές μελλοντικές επεκτάσεις περιλαμβάνουν επεκτάσεις του μηχανισμού και εκτενέστερη εξερεύνηση των πολιτικών χρήσης του. Συγκεκριμένα, ο μηχανισμός μπορεί να επεκταθεί ώστε να υποστηρίξει εικονικά μηχανήματα με περισσότερες από μία vcpus. Επιπλέον, ο μηχανισμός μπορεί να αποφύγει να καταστρέφει τα EPTs, που είναι σχετικά στατικά, όταν γίνεται η αλλαγή από την μία μέθοδο σελιδοποίησης στην άλλη. Αυτό απαιτεί την παρακολούθηση και την εφαρμογή των αλλαγών στις μεταφράσεις της εφαρμογής που υλοποιεί το εικονικό μηχάνημα στα EPTs, ώστε να παραμείνουν συγχρονισμένα.

Τέλος, σχετικά με τις πολιτικές, θα μπορούσε να γίνει πειραματισμός και για τις υπόλοιπες τιμές των παραμέτρων που υπάρχουν, όπως επίσης και να δοκιμαστεί η επίδοση του μηχανισμού σε benchmarks που κάνουν μεγαλύτερη χρήση της μνήμης, χρησιμοποιώντας παράλληλα και hugepages.

Βιβλιογραφία

[Bae11] Chang S. Bae, John R. Lange and Peter A. Dinda, "Enhancing Virtualized Application Performance through Dynamic Adaptive Paging Mode Selection", in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, p. 255–264, New York, NY, USA, 2011, Association for Computing Machinery.

[Gand14] Jayneel Gandhi, Arkaprava Basu, Mark D. Hill and Michael M. Swift, "Efficient Memory Virtualization: Reducing Dimensionality of Nested Page Walks", in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, p. 178–189, USA, 2014, IEEE Computer Society.

[Gand16] Jayneel Gandhi, Mark Hill and Michael Swift, "Agile Paging: Exceeding the Best of Nested and Shadow Paging", *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 707–718, 06 2016.

[Hieb18] Jason Hiebel, Laura E. Brown and Zhenlin Wang, "Constructing Dynamic Policies for Paging Mode Selection", in *Proceedings of the 47th International Conference on Parallel Processing*, ICPP 2018, New York, NY, USA, 2018, Association for Computing Machinery.

[Hoan10] Giang Hoang, Chang Bae, John Lange, Lide Zhang, Peter Dinda and Russ Joseph, "A Case for Alternative Nested Paging Models for Virtualized Systems", *Computer Architecture Letters*, vol. 9, pp. 17–20, 01 2010.

[Inte] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual*, Intel Corporation.

[Kara15] Vasileios Karakostas, Jayneel Gandhi, Furkan Ayar, Adrián Cristal, Mark D. Hill, Kathryn S. McKinley, Mario Nemirovsky, Michael M. Swift and Osman Ünsal, "Redundant Memory Mappings for Fast Access to Large Memories", in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, p. 66–78, New York, NY, USA, 2015, Association for Computing Machinery.

[Kuan14] Wei Kuang, Laura E. Brown and Zhenlin Wang, "Selective switching mechanism in virtual machines via support vector machines and transfer learning", *Machine Learning*, vol. 101, pp. 137–161, 2014.

[LWN08] LWN, "Memory management notifiers", 2008. https://lwn.net/Articles/266320/.

[LWN11] LWN, "Transparent huge pages in 2.6.38", 2011. https://lwn.net/Articles/423584/.

[LWN17a] LWN, "The current state of kernel page-table isolation", 2017. https://lwn.net/Articles/741878/.

[LWN17b] LWN, "Five-level page tables", 2017. https://lwn.net/Articles/717293/.

[Marg18] Artemiy Margaritov, Dmitrii Ustiugov, Edouard Bugnion and Boris Grot, "Virtual Address Translation via Learned Page Table Indexes", in *Proceedings of the Workshop on ML for Systems at NeurIPS co-located with the 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, 2018. Workshop on ML for Systems at NeurIPS , NIPS 2018 ; Conference date: 08-12-2018 Through 08-12-2018.

[Pham15] Binh Pham, Ján Veselý, Gabriel H. Loh and Abhishek Bhattacharjee, "Large Pages and Lightweight Memory Management in Virtualized Environments: Can You Have It Both Ways?", in *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, p. 1–12, New York, NY, USA, 2015, Association for Computing Machinery.

[Qumr07] Avi Qumranet, Yaniv Qumranet, Dor Qumranet, Uri Qumranet and Anthony Liguori, "KVM: The Linux virtual machine monitor", *Proceedings Linux Symposium*, vol. 15, 01 2007.

[Wang11] Xiaolin Wang, Jiarui Zang, Zhenlin Wang, Yingwei Luo and Xiaoming Li, "Selective Hardware/Software Memory Virtualization", in *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, p. 217–226, New York, NY, USA, 2011, Association for Computing Machinery.

[Yani16] Idan Yaniv and Dan Tsafrir, "Hash, Don't Cache (the Page Table)", *SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, p. 337–350, June 2016.

[Zhan17] Yu Lin Zhang, Peter Tröger and Matthias Werner, "Dynamic Paging Method Switching - An Implementation for KVM", in *ISC Workshops*, 2017.