ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Σχεδίαση και Υλοποίηση Μηχανισμού Διαχείρισης Ελαστικής Μνήμης σε Εικονικά Περιβάλλοντα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αιμίλιου Τσαλαπάτη

Επιβλέπων: Γεώργιος Γκούμας
Επίκουρος Καθηγητής

Αθήνα, Μάϊος 2018

Εθνικο Μετσοβιο Πολυτεχνειο
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Σχεδίαση και Υλοποίηση Μηχανισμού Διαχείρισης Ελαστικής Μνήμης σε Εικονικά Περιβάλλοντα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αιμίλιου Τσαλαπάτη

Επιβλέπων: Γεώργιος Γκούμας
Επίκουρος Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Μαΐου.

.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Νικόλαος Παπασπύρου
Επίκουρος Καθηγητής	Καθηγητής	Αναπληρωτής Καθηγητής

Αθήνα, Μάϊος 2018.

.....

Αιμίλιος Τσαλαπάτης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

Περίληψη

Σε περιβάλλοντα στα οποία πολλαπλές εικονικές μηχανές εκτελούνται πάνω στο ίδιο φυσικό μηχάνημα, η αντιμετώπισή των guests ως απλές διεργασίες από το host οδηγεί στη χαμηλότερη απόδοση του συστήματος όσον αφορά την κατανομή της μνήμης μεταξύ του λειτουργικού συστήματος του host και των εικονικών μηχανών. Σε αυτήν την περίπτωση, λύσεις όπως το ballooning, οι οποίες βασίζονται στην εκτίμηση της ποσότητας καταναλισκώμενης μνήμης από το guest, είναι ανίκανες να μεταβάλλουν την κατανομή της μνήμης αρκετά γρήγορα για να αποφύγουν παροδικές ελλείψεις μνήμης στο guest[1]. Εναλλακτικές όπως το Transcendent Memory λειτουργούν πάνω στο υποσύστημα διαχείρισης μνήμης του guest, επιτρέποντάς του να χρησιμοποιήσει μνήμη του host αντί να χρειάζεται να καταφύγει σε εναλλαγή σελίδων στο δίσκο(swapping). Όλες αυτές οι τεχνικές λειτουργούν στο επίπεδο του υποσυστήματος διαχείρισης μνήμης, με αποτέλεσμα να μπορούν να χρησιμοποιηθούν μόνο έμμεσα από τις εφαρμογές. Ως αποτέλεσμα, υπάρχουν περιπτώσεις όπου απότομες μεταβολές στην κατανάλωση μνήμης οδηγούν στο λειτουργικό σύστημα του guest να μην μπορεί να εκμεταλλευθεί τους μηχανισμούς αυτούς, και να καταφεύγει σε εναλλαγή, ή ακόμα και στον OOM killer. Σε αυτήν την εργασία προτείνουμε το μηχανισμό utmem (Userspace Transcendent Memory), μία έκδοση του Transcendent Memory η οποία μπορεί να χρησιμοποιηθεί απέυθείας από τις εφαρμογές, χωρίς την παρεμβολή του λειτουργικού συστήματος του guest. Οι μετρήσεις μας αποδεικνύουν ότι η προσέγγισή μας επιτρέπει στους guests να μεταβάλλουν γρήγορα την ποσότητα καταναλισκώμενης μνήμης, χωρίς την πτώση σε απόδοση η οποία προκαλείται από την απότομη εφαρμογή του μηχανισμού ballooning.

Λέξεις Κλειδιά: Εικονοποίηση, Διαχείριση Μνήμης, Υπολογισμός Νέφους

Abstract

In environments where multiple virtual machines are colocated on the same physical host, the treatment of the guests as ordinary processes by the host leads to the suboptimal distribution of memory between the host and the guests. In this case, solutions such as ballooning, based on working set estimation, are unable to adapt memory distributions fast enough to avoid temporarily starving a guest out of memory[1]. Alternatives such as Transcendent Memory work by being bolted on the guest's swap subsystem, and allowing it to use host memory instead of resorting to swapping. All these techniques are applied at the memory management subsystem level, therefore being only indirectly leveraged by applications; this results in cases where the sharp fluctuations in memory utilization result in the guest OS being unable to take advantage of them, resulting in unneeded swapping, and even the OOM killer terminating the workload. We propose Userspace Transcendent Memory (utmem), a version of Transcendent Memory that can be directly utilized by applications without interference by the OS. Our results demonstrate that such an approach succeeds in allowing the guests to rapidly fluctuate their memory usage without the adverse effects associated with rapid ballooning.

Keywords: Virtualization, Memory Management, Cloud Computing

Ευχαριστίες

Η διπλωματική εργασία αυτή εκπονήθηκε το ακαδημαϊκό έτος 2017-2018 στο Εργαστήριο Υπολογιστικών Συστημάτων του Εθνικού Μετσόβιου Πολυτεχνείου. Θα ήθελα να ευχαριστήσω θερμά τους Στέφανο Γεράγγελο, Τάσσο Νάνο, Στράτο Ψωμαδάκη, και Κωστή Παπαζαφειρόπουλο, με τους οποίους είχα μια εξαιρετική συνεργασία κατά τα 3 χρόνια στα οποία συμμετείχα στο εργαστήριο. Χωρίς τη σταθερή καθοδήγηση τους και τις καθοριστικής σημασίας συμβουλές τους, η πλήρωση της παρούσας εργασίας θα ήταν αδύνατη.

Αιμίλιος Τσαλαπάτης
Αθήνα, 19 Μαΐου 2018

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Τι είναι η εικονοποίηση;

Ο όρος εικονοποίηση (virtualization) αναφέρεται στην απεικόνιση ενός υπολογιστικού πόρου με τέτοιο τρόπο ώστε ο καταναλωτής της (αναφερόμενος ως guest) συμπεριφέρεται με τον ίδιο τρόπο όπως αν το διαχειριζόταν ο ίδιος, ενώ στην πραγματικότητα η διανομή του γίνεται από κάποιο άλλο κομμάτι λογισμικού (το οποίο ονομάζεται host). Αποτελεί διαφορετική τεχνική από την προσομοίωση (emulation), όπου ένα πρόγραμμα το οποίο λέγεται προσομοιωτής αναπαράγει τη συμπεριφορά κάποιου προγράμματος ή και εξαρτήματος υλικού. Τέτοιου είδους προγράμματα, όπως το QEMU (Quick emulator) [2], μπορούν να προσομοιώσουν και οντότητες οι οποίες δεν υπάρχουν στο σύστημα, όπως εξειδικευμένες μνήμες ή κάρτες δικτύου. Η εικονοποίηση μπορεί να υλοποιηθεί σε διάφορα επίπεδα πάνω στη στοίβα λογισμικού, όπως για παράδειγμα στο επίπεδο εφαρμογών, όταν χρησιμοποιούνται εικονικές μηχανές γλωσσών (π.χ. το Java Virtual Machine, JVM) [3]. Εναλλακτικά, η εικονοποίηση παρέχεται στο επίπεδο λειτουργικού συστήματος, όταν χρησιμοποιείται το λεγόμενο containerization. Εκεί, εικονικοποιημένες εκδόσεις των πόρων που παρέχονται από το λειτουργικό, όπως network sockets ή process identifiers, PIDs, δίνονται στις εφαρμογές μέσα από λογισμικό όπως το docker[4]. Τρίτη επιλογή είναι η εικονοποίηση στο επίπεδο συστήματος, όπου το ίδιο το υλικό (επεξεργαστής, φυσική μνήμη» εικονοποιείται μέσα από τη χρήση λογισμικού hypervisor όπως τα KVM[5] και Xen[6]. Σε αυτό το κείμενο, με τον όρο εικονοποίηση θα αναφερόμαστε πάντα στην εικονοποίηση επιπέδου συστήματος.

1.2 Πριν την εικονοποίηση

Ένα από τα παλαιότερα προβλήματα στο πεδίο των λειτουργικών συστημάτων αποτελεί η διαχείριση και διαμοίραση πόρων σε ένα μοιραζόμενο σύστημα. Το ζήτημα αυτό προέκυψε τη δεκαετία του ΄60, καθώς πριν από τότε ο καθιερωμένος τρόπος χρήσης των υπολογιστών (οι οποίοι ήταν όλοι mainframes) ήταν το λεγόμενο batch

computing. Σύμφωνα με το σύστημα αυτό, ο χρήστης έπρεπε να χρησιμοποιήσει χάρτινες κάρτες στις οποίες είχε κωδικοποιήσει το πρόγραμμά του (punch cards), και να τις παραδόσει για εκτέλεση σε μια ουρά εργασιών. Αφότου έφτανε η σειρά του, το πρόγραμμα εκτελούταν, και ο χρήστης ενημερωνόταν για το αποτέλεσμα. Με την επάνοδο του συστήματος του διαδραστικού προγραμματισμού, στο οποίο οι χρήστες χρησιμοποιούσαν τερματικά για να στέλνουν και να παραλαμβάνουν δεδομένα από τον υπολογιστή ενώ αυτός έτρεχε, η ταυτόχρονη χρήση των υπολογιστών έγινε εφικτή. Η εξέλιξη αυτή κατέστησε απαραίτητη τη δημιουργία μηχανισμών με σκοπό τον ορθό και δίκαιο διαμοιρασμό των πόρων στους χρήστες, έτσι ώστε η εκτέλεση των προγραμμάτων του ενός να μην επηρεάζουν αυτήν των άλλων, καθώς και να έχουν όλοι αρκετούς υπολογιστικούς πόρους (μνήμη, κύκλους επεξεργαστή) ώστε να μπορούν να εκτελέσουν ουσιαστικές εργασίες.

Ως πιθανές λύσεις του προβλήματος του διαμοιρασμού προέκυψαν δύο πιθανά μοντέλα συστήματος. Το ένα υιοθετήθηκε από το λειτουργικό σύστημα Unix, και βασιζόταν στη δημιουργία πολλαπλών λογαριασμών στο λειτουργικό σύστημα, καθένας από τους οποίους αντιστοιχούσε σε ένα χρήστη. Με βάση αυτό το διαμοιρασμό, το λειτουργικό τότε εφάρμοζε πολιτικές κατανομής των πόρων κεντρικά. Το μοντέλο αυτό, αν και επιτυγχάνει το επιθυμητό αποτέλεσμα θεωρητικά, είναι απαραίτητο να υλοποιηθεί πολύ προσεκτικά έτσι ώστε να περιορίζει πραγματικά το χρήστη στο δικό του χώρο. Αν η υλοποίηση εμπεριέχει σφάλματα, ο ένας λογαριασμός μπορεί να υπερβαίνει τα όριά του, τόσο ως προς τα δεδομένα στα οποία έχει πρόσβαση όσο και στους πόρους οι οποίοι του έχουν ανατεθεί. Η σταθερότητα του συστήματος συνολικά είναι επίσης μειωμένη, καθώς αν μία διαδικασία χρήστη τερματιστεί ανώμαλα (crashes) μπορεί να προκαλέσει και crash σε όλο το σύστημα.

Η δεύτερη προσέγγιση του προβλήματος υλοποιήθηκε σε ένα λειτουργικό υλοποιημένο από την IBM, το CP/CMS. Το λειτουργικό αυτό αποτελούταν από δύο μέρη: Το Control Program (CP) ήταν υπεύθυνο για τη διαχείριση των πόρων του συστήματος, ενώ το Console Monitoring System (CMS) ήταν ένα μικρό λειτουργικό σύστημα με μοναδικό χρήστη (single-user), το οποίο και αποτελούσε τον τρόπο επικοινωνίας του τελευταίου με το σύστημα μέσω ενός τερματικού. Κατάυτόν τον τρόπο, ο έλεγχος πρόσβασης και ο διαμοιρασμός πόρων αποτελούσαν εγγενείς ιδιότητες του συστήματος, αφού το κάθε CMS εξέτειθε ένα σύνολο πόρων στο χρήστη διαφορετικό από αυτό του συνολικού συστήματος. Επιπλέον, το σύστημα ήταν πιο σταθερό, αφού ένα crash στο CMS δεν επηρέαζε το CP. Το σύστημα αυτό αποτελεί ένα από τα πρώτα παραδείγματα εικονοποίησης, αφού βασιζόταν στην παρουσίαση στο χρήστη ακριβώς των πόρων στους οποίους είχαν πρόσβαση. Δεν επέτρεπε τη θεώρηση του συστήματος στο σύνολό του, επιτυγχάνοντας την απομόνωση του ενός χρήστη από τον άλλον.

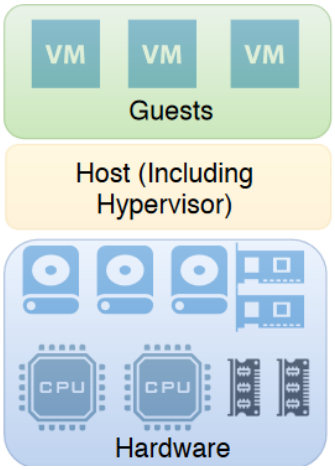
Άλλη μία ιδέα η οποία έχει τις ρίζες της σε εκείνη την εποχή, και η οποία χρησιμοποιείται για την επίλυση του προβλήματος της απομόνωσης του χρήστη, είναι αυτή της εικονικής μνήμης (virtual memory, VM). Η εικονική μνήμη αποτελεί μια αφηρημένη

αναπαράσταση της μνήμης ενός συστήματος, η οποία αντιστοιχίζεται σε φυσική μνήμη με τη βοήθεια του υλικού. Σε ένα σύστημα το οποίο υποστηρίζει εικονική μνήμη, οι διεργασίες οι οποίες τρέχουν σχεδόν ποτέ δε γνωρίζουν τη φυσική μνήμη την οποία χρησιμοποιούν, και αντάυτού εκτελούνται μέσα σε ένα χώρο εικονικών διευθύνσεων στον οποίον μπορούν να αποθηκεύσουν δεδομένα. Μέσα από τη χρήση αυτής της μεθόδου, οι διεργασίες δε χρειάζεται να πάρουν άλλα μέτρα για να προστατεύσουν τα δεδομένα τους από τις υπόλοιπες. Ένα εκτελέσιμο δεν μπορεί να καταστρέψει τα δεδομένα ενός άλλου, αφού δεν υπάρχει διεύθυνση που αντιστοιχεί σε αυτά στο χώρο διευθύνσεών του. Το στρώμα αφαίρεσης αυτό παρέχει και τη δυνατότητα απλούστερων μηχανισμών διαχείρισης μνήμης: Αφού μία διεργασία εργάζεται μόνο με εικονικές διευθύνσεις, το λειτουργικό σύστημα μπορεί να μεταφέρει τα δεδομένα σε άλλα σημεία της φυσικής μνήμης, ή ακόμη και να τα στείλει στο δίσκο με μία διαδικασία η οποία λέγεται εναλλαγή (swapping). Ακόμα ένα πλεονέκτημα είναι πως, αφού τα εκτελέσιμα λειτουργούν με εικονικές διευθύνσεις, μπορούν να φορτωθούν οπουδήποτε στη μνήμη για να εκτελεστούν.

Και οι δύο τεχνολογίες αυτές αποτελούν παραδείγματα τεχνικών εικονοποίησης, δηλαδή τεχνικών οι οποίες παρέχουν μια αφηρημένη έκδοση ενός υπολογιστικού πόρου στον καταναλωτή του, με σκοπό τη διευκόλυνση της διαχείρισης του πρώτου από ένα πρόγραμμα ελέγχου. Οι τεχνικές αυτές επεκτείνονται με τη μετατροπή των προγραμμάτων ώστε να γνωρίζουν και να εκμεταλλεύονται το μηχανισμό αυτό, συνεργαζόμενα με το λειτουργικό με σκοπό την ακόμα πιο εύκολη διαχείριση. Συχνή σχεδιαστική επιλογή σε τέτοια συστήματα είναι η επιτάχυνση της εικονοποίησης μέσα από τη χρήση εξειδικευμένου υλικού για τη γρήγορη εκτέλεση εργασιών οι οποίες θα ήταν περίπλοκες ή χρονοβόρες αν βασίζονταν αποκλειστικά σε λογισμικό. Το πιο αντιπροσωπευτικό παράδειγμα τέτοιου είδους υλικού είναι το Memory Management Unit (MMU), του οποίου ο σκοπός είναι η επιτάχυνση των μεταφράσεων των εικονικών διευθύνσεων σε φυσικές. Καθώς το να ανατρέχει ο επεξεργαστής στη μνήμη για κάθε πρόσβαση στη μνήμη θα ήταν εξαιρετικά αργό, τα μοντέρνα συστήματα έχουν επίσης υλικό το οποίο αποκαλείται Translation Lookaside Buffer (TLB), το οποίο είναι μία κρυφή μνήμη (cache) για αντιστοιχίες εικονικών σε φυσικές διευθύνσεις. Καθώς το TLB αποθηκεύει τα πιο συχνά χρησιμοποιούμενα ζεύγη εικονικών-φυσικών μνημών, επιταχύνει τη μετάφραση.

1.3 Εικονοποίηση

Αν και η παρουσίαση στις διεργασίες ενός συστήματος ενός αφηρημένου μοντέλου του τελευταίου έχει αποτελέσει ευρύτατα χρησιμοποιούμενη τεχνική από τότε που κατασκευάστηκαν οι παραπάνω μηχανισμοί, η εικονοποίηση ολόκληρων συστημάτων δεν υιοθετήθηκε ως τεχνική, με τα περισσότερα λειτουργικά να ακολουθούν το παράδειγμα του Unix. Μόνο στα τέλη της δεκαετίας του 1990 και στις αρχές αυτής του 2000 επανήλθε το ενδιαφέρον στην ιδέα αυτή, με τη δημιουργία εποπτών (hypervisors) όπως το VMWare Workstation (1999) και του Xen (το οποίο εκδόθηκε ως ανοικτό



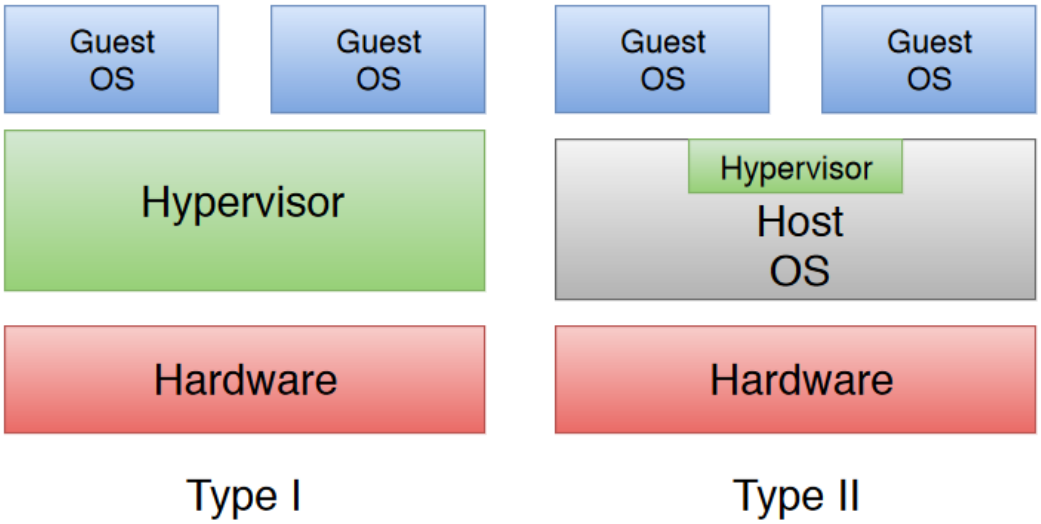
Σχήμα 1.1: Επισκόπηση ενός εικονοποιημένου συστήματος.

λογισμικό το 2002). Η υποστήριξη των εποπτών από υλικό έγινε εφικτή με το σχεδιασμό επεκτάσεων στο σύνολο εντολών x86, οι οποίες είναι οι VT-x για επεξεργαστές Intel και AMD SVM για AMD. Κατάυτόν τον τρόπο, η εικονοποίηση επιταχύνθηκε αρκετά για να αποτελεί αποδοτική τεχνική, χρήσιμη στο σχεδιασμό ενός συστήματος.

1.3.1 Επόπτες τύπου I και τύπου II

Το πρόγραμμα επόπτη αποτελεί κεντρικό κομμάτι κάθε συστήματος που χρησιμοποιεί εικονοποίηση. Οι επόπτες εκεί έχουν ένα ρόλο παρεμφερή με αυτόν του λειτουργικού συστήματος στα συμβατικά περιβάλλοντα. Ρόλος του είναι να εκτελούν εκ μέρους των χρηστών του (τις εικονικές μηχανές, δηλαδή), ¨προνομιούχες' ενέργειες οι οποίες είναι δυνατές μόνο από τα προγράμματα-διαχειριστές του συστήματος (privileged operations).

Υπάρχουν δύο διαφορετικά είδη εποπτών, ανάλογα με το που βρίσκονται στο σύστημα. Οι επόπτες τύπου I (type I/bare metal hypervisors) εκτελούνται 'πάνω στο υλικό', χωρίς να βασίζονται σε κάποιο άλλο λειτουργικό. Το πιο γνωστό παράδειγμα τέτοιου επόπτη είναι ο Xen. Προφανώς, καθώς δεν υπάρχει λειτουργικό ανάμεσα στον επόπτη και το υλικό, τίθεται το θέμα του τρόπου διαχείρισης του τελευταίου - η συντριπτική πλειοψηφία του κώδικα των μοντέρνων λειτουργικών συστημάτων αποτελείται από οδηγούς υλικού, και η αναπαραγωγή της λειτουργικότητας αυτής στον επόπτη δε θα ήταν πρακτική. Ο Xen αντιμετωπίζει το θέμα αυτό αναθέτοντας το ρόλο της 'εικονικής μηχανής ελέγχου' (dom0) σε κάποιο από τις εκτελούμενες εικονικές μηχανές, και επιτρέποντάς της να διαχειρίζεται το υλικό εκ μέρους του. Οι υπόλοιπες μηχανές είναι'εικονικές μηχανές χρήστη' (domU), και βασίζονται στον επόπτη και τη μηχανή ελέγχου για την εκτέλεση προνομιούχων ενεργειών όπως η διαχείριση του εικονοποιημένου υλικού τους. Η προσομοίωση συσκευών, όταν αυτό είναι απαρα-



Σχήμα 1.2: Επισκόπηση συστημάτων με επόπτες τύπου I και τύπου II.

ίτητο, γίνεται στο dom0 χρησιμοποιώντας το QEMU [2], το οποίο είναι δυνατόν να προσομοιώσει συγκεκριμένες συσκευές ή και ολόκληρα συστήματα.

Το άλλο πιθανό είδος επόπτη είναι αυτό του τύπου II (type II/hosted hypervisor). Αυτό είναι σχεδιασμένο ως κομμάτι ενός συμβατικού λειτουργικού συστήματος. Από αυτού το είδους τους επόπτες ο πιο διαδεδομένος είναι ο KVM, ο οποίος αρχικά ήταν εφικτό να χρησιμοποιηθεί μόνο με το λειτουργικό σύστημα Linux, όμως μεταφέρθηκε και σε πληθώρα άλλων. Άλλοι επόπτες τύπου II είναι ο bhyve για το FreeBSD, και ο Hyper-V, που είναι κομμάτι των Windows. Στα συστήματα που χρησιμοποιούνται, το λειτουργικό του host είναι αυτό το οποίο οφείλει να προσομοιώνει το εικονοποιημένο λογισμικό (με τη βοήθεια προσομοιωτή). Ως αποτέλεσμα, ο επόπτης στην περίπτωση αυτή έχει το ρόλο του ενδιάμεσου μεταξύ guest και host, ελέγχοντας τα αιτήματα για εκτέλεση προνομιούχων ενεργειών ώστε να επιβεβαιώσει πως επιτρέπονται, πριν τις προωθήσει στο κατάλληλο λογισμικό. Επικεντρώνοντας την προσοχή μας στον KVM, οι guests εκτελούνται ως διεργασίες QEMU, όμως την περισσότερη ώρα χρησιμοποιούν τον επόπτη για να τρέχουν, και βγαίνουν από αυτόν μόνο για να προσομοιώσουν υλικό πριν συνεχίσουν την εκτέλεση. Αξίζει να σημειωθεί πως αυτού του είδους οι επόπτες τρέχουν αποκλειστικά με τη βοήθεια επεκτάσεων εικονοποίησης του επεξεργαστή - αυτές αναλύονται στην επόμενη ενότητα.

1.3.2 Επεκτάσεις εικονοποίησης

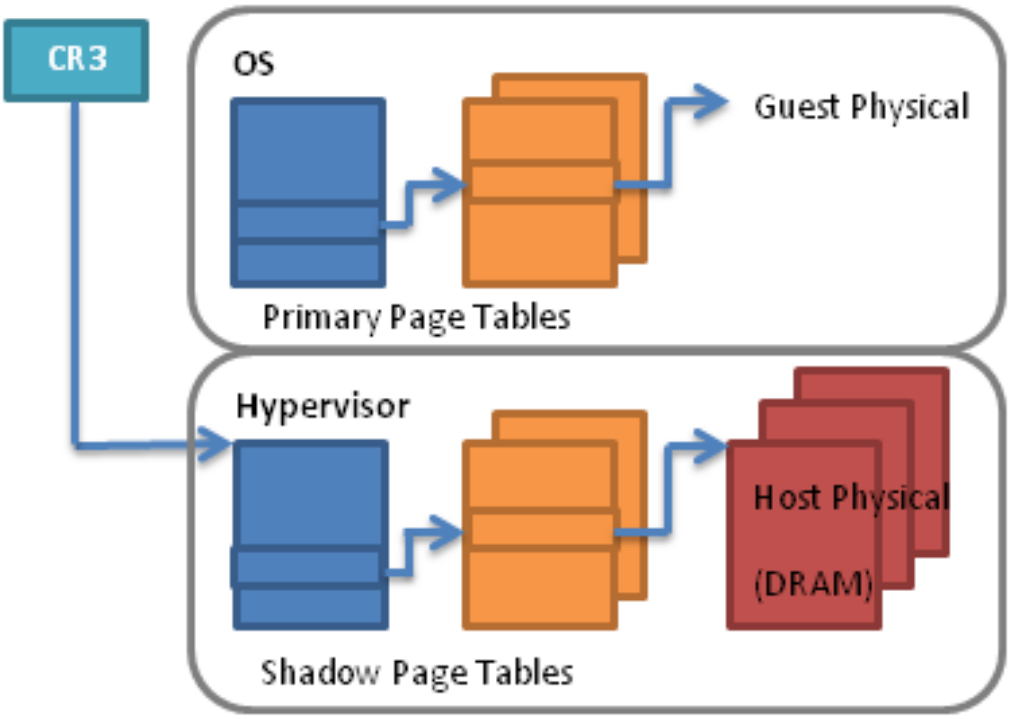
Έως τώρα έχει αναφερθεί σε διάφορα σημεία του κειμένα η χρήση επεκτάσεων εικονοποίησεις, οι οποίες είναι επεκτάσεις υλικού - πιο συγκεκριμένα, του επεξεργαστή - με σκοπό την επιτάχυνση της διαδικασίας της εικονοποίησης. Η ιδέα αυτή αποδείχτηκε

κομβικής σημασίας για την ανάπτυξη των υλοπογιστικών νεφών (computing clouds), και του cloud computing γενικότερα. Πρόκειται στην ουσία για το σχεδιασμό επεξεργαστών οι οποίοι έχουν επίγνωση πως ακόμα και τα πιο βασικά τους στοιχεία (π.χ. καταχωρητές ελέγχου), καθώς και τα σήματα τα οποία παράγει (π.χ. διακοπές χρονιστών, timer interrupts) έχουν πολλαπλούς πιθανούς καταναλωτές (τους guests). Αυτή η πολύπλεξη της εκτέλεσης πολλαπλών λειτουργικών πάνω στο ίδιο υλικό έρχεται σε πλήρη αντίθεση με την αρχιτεκτονική των συμβατικών (έως τα μέσα της δεκαετίας του 2000) συστημάτων, τα οποία είχαν κατασκευαστεί υποθέτοντας πως μόνο ένα λειτουργικό θα εκτελούταν ανά πάσα στιγμή.

Οι επεκτάσεις εικονοποίησης οι οποίες αναφέρονται στην αρχιτεκτονική x86 είναι, όπως αναφέρθηκε, οι SVM για την AMD και VT-x για την Intel. Από εδώ και στο εξής, θα κάνουμε χρήση της ορολογίας του δεύτερου. Οι επεκτάσεις αυτές χρησιμοποιούνται, λοιπόν, εκτελώντας μία ειδική εντολή με το όνομα VMXON, ώστε να ενεργοποιηθεί και το υπόλοιπο του συνόλου εντολών. Όταν, ύστερα, μία εικονική μηχανή ξεκινά να εκτελείται, ο επεξεργαστής εκτελεί μία εντολή VMLAUNCH, η οποία αρχικοποιεί κάποιες ειδικές δομές δεδομένων οι οποίες χρειάζονται για την αποθήκευση της κατάστασης της εικονικής μηχανής σε διάφορα σημεία του κύκλου ζωής της. Κατά τη διάρκεια του τελευταίου, ο επεξεργαστής εκτελεί μία εντολή VMEXIT για να σταματήσει προσωρινά την εκτέλεση, και να δώσει στο host τη δυνατότητα να διαχειριστεί κάποια προνομιούχα εντολή η οποία επιχειρήθηκε να εκτελεστεί στο guest. Όταν αυτό με τη σειρά του επιτευχθεί, εκτελείται στον επεξεργαστή η εντολή VMRESUME, ώστε να επανέλθει η ροή εκτέλεσης στην εικονική μηχανή. Τέλος, όταν δεν είναι απαραίτητες οι ειδικές εντολές αυτές, ο επεξεργαστής εκτελεί για να τις απενεργοποιήσει την εντολή VMXOFF.

Ο τρόπος αυτός διαχείρισης των εικονικών μηχανών παρουσιάζει πληθώρα πλεονεκτημάτων: Για παράδειγμα, δομές δεδομένων του επεξεργαστή οι οποίες είναι προσβάσιμες μόνο με προνομιούχες εντολές μπορούν να χρησιμοποιηθούν κατά την εκτέλεση του guest. Η πιο διαδεδομένη χρήση της δυνατότητας αυτής λαμβάνει χώρα κατά τη διαχείριση του καταλόγου σελίδων (page table) στο guest. Όταν δεν υπάρχει υποστήριξη από το υλικό, όπως για παράδειγμα όταν χρησιμοποιείται αποκλειστικά παραεικονοποίηση (paravirtualization, περισσότερα πάνω σε αυτό παρακάτω), ο host χρειάζεται να διατηρεί σκιώδεις καταλόγους σελίδων (shadow page tables), οι οποίοι να διατηρούν αντιστοιχίσεις φυσικών διευθύνσεων guest με φυσικές διευθύνσεις host. Έτσι, κάθε αλλαγή αντιστοίχισης μέσα στον πρώτο χρειάζεται να ακολουθείται από άλλη μία στο δεύτερο. Το τελικό αποτέλεσμα αυτής της μεθόδου είναι το αυξημένο κόστος των μετατροπών των καταλόγων σελίδων, αφού κάθε μία από αυτές συνοδεύεται από έξοδο από την εικονική μηχανή.

Η εναλλακτική στο μηχανισμό αυτό, είναι τα Extended Page Tables (EPT) (ή Nested Page Tables (NPT) για AMD επεξεργαστές). Αυτά αποτελούν μία στοίβα καταλόγων σελίδων, με τα χαμηλότερα επίπεδα να χρησιμοποιούν τα αποτελέσματα



Σχήμα 1.3: Σκιώδεις κατάλογοι σελίδων στον επόπτη. "ShadowPageTables" is licensed under CC SA 4.0

.

των μεταφράσεων από τα υψηλότερα. Έτσι, για κάθε είσοδο στο μηχανισμό αυτό εκτελείται αυτόματα μία αλυσίδα μεταφράσεων, μία σε κάθε επίπεδο. Η δομή δεδομένων αυτή διατρέχεται αυτόματα από το υλικό, χωρίς να χρειάζεται έξοδος από την εικονική μηχανή. Από τη μία πλευρά, η χρήση των καταλόγων σελίδων είναι πιο χρονοβόρα λόγω των επιπλέον στρωμάτων μετάφρασης, και άρα τα TLB misses κοστίζουν περισσότερο [7]. Από την άλλη πλευρά, οι μετατροπές των καταλόγων είναι πολύ πιο γρήγορες, αφού δε χρειάζονται τη διατήρηση δομών όπως σκιώδεις καταλόγους.

1.3.3 Είδη εικονοποίησης

Όπως προαναφέρθηκε, υπάρχουν διαφορετικές μέθοδοι οι οποίες μπορούν να εφαρμοστούν για να επιτευχθεί η εικονοποίηση επιπέδου συστήματος. Οι τεχνικές αυτές βασίζονται τόσο στη δυνατότητα που προσφέρει το υλικό να επιταχύνει διαδικασίες απαραίτητες για την υποστήριξη εκτέλεσης guests, όσο και στην άμεση συνεργασία του guest με το host με σκοπό την επίτευξη γρηγορότερης και απλούστερης εκτέλεσης [8]. Η τελευταία αυτή τεχνική είναι δυνατή μόνο όταν ο guest αναγνωρίζει πως δεν τρέχει

Πίνακας 1.1: Είδη εικονοποίησης επιπέδου συστήματος

Support	*Guest*	*Hardware*
Όχι	Πλήρης Εικονοποίηση	Εικονοποίηση Υποστηριζόμενη από Υλικό
Ναι	Παραεικονοποίηση	Μικτή Εικονοποίηση

πάνω στο υλικό απέυθείας, αλλά παρεμβάλλεται τουλάχιστον άλλο ένα λειτουργικό ανάμεσα, και συνεργάζεται με αυτό. Παρακάτω δίνεται μία επισκόπηση των διαφόρων ειδών εικονοποίησης, τοποθετημένα με τη σειρά με την οποία αναπτύχθηκαν:

- Εικονοποίηση πλήρους συστήματος: Ο guest δεν έχει υποστεί καμία αλλαγή (τρέχει δηλαδή ένα συμβατικό λειτουργικό), ενώ και το υλικό το οποίο χρησιμοποιείται δεν έχει επεκτάσεις εικονοποίησης. Σε αυτήν την περίπτωση, το αν ο guest μπορεί να τρέξει ως έχει εξαρτάται από την αρχιτεκτονική του επεξεργαστή του συστήματος. Πιο συγκεκριμένα, υπάρχουν σύνολα εντολών τα οποία δε χρειάζονται επεκτάσεις ώστε να μπορούν να τρέξουν εικονικές μηχανές με αποδοτικό τρόπο. Τα σύνολα αυτά ικανοποιούν τις προϋποθέσεις οι οποίες διατυπώθηκαν από τους Popek και Goldberg [9]. Σε περίπτωση όπου οι τελευταίες ικανοποιούνται, ο host χρειάζεται μόνο να παγιδεύει (trap) τις προνομιούχες εντολές οι οποίες εκτελούνται στο guest, και να προσομοιώνει κατάλληλα τις επιθυμητές ενέργειες. Στην περίπτωση της αρχιτεκτονικής x86, οι συνθήκες αυτές δυστυχώς δεν ικανοποιούνται, οπότε η πλήρης εικονοποίηση είναι αδύνατη. Εφικτή είναι μόνο η προσομοίωση, η οποία και εκτελείται από προγράμματα όπως το QEMU. Αυτό επιτυγχάνει το στόχο του μεταφράζοντας το δυαδικό εκτελέσιμο κατάλληλα ώστε να αφαιρεί προβληματικές και προνομιούχες εντολές, και να τις αντικαθιστά με κώδικα που εκτελεί τις ίδιες λειτουργίες σε προσομοιωμένο υλικό. Προσομοιωμένες είναι επίσης και όλες οι συσκευές τις οποίες νομίζει ο guest ότι διαχειρίζεται, με το πρόγραμμα να λαμβάνει τις εντολές και να τις μετατρέπει σε ενέργειες πάνω σε ένα μοντέλο της συσκευής που παρουσιάζεται στην 'εικονική μηχανή'.

- Παραεικονοποίηση (Paravirtualization): Στην περίπτωση αυτή, ενώ το υλικό και πάλι δεν μπορεί να χρησιμοποιηθεί για να επιταχύνει την εικονοποίηση, το λειτουργικό σύστημα του guest γνωρίζει πως τρέχει πάνω από άλλο λειτουργικό σύστημα, και μεταβάλλει τη συμπεριφορά του ανάλογα. Εκτελώντας αντί προνομιούχων εντολών ενέργειες οι οποίες έχουν 'συμφωνηθεί' να αποτελούν σήματα στο host για να εκτελέσει κάποια λειτουργία, όπως π.χ. η εγγραφή μιας συγκεκριμένης θέσης μνήμης για να παραλάβει ο host δεδομένα από μια ουρά σε δεδομένο σημείο της φυσικής μνήμης του guest, ο τελευταίος διευκολύνει τον πρώτο παραλείποντας εντολές οι οποίες θα έπρεπε να προσομοιωθούν. Για

παράδειγμα, χάρη στη συνεργασία αυτή ο host μπορεί να προσομοιώνει αφηρημένες αντί για πραγματικές συσκευές, με απλούστερη διαδικασία προσομοίωσης και ελέγχου. Η επικοινώνία μεταξύ των δύο μερών του συστήματος είναι λοιπόν ταχύτερη. Χαρακτηριστικά παραδείγματα αυτού του είδους εικονοποίησης είναι οι συσκευές virtio, κλάση 'υλικού' που περιλαμβάνει από συσκευές μπλοκ έως κάρτες δικτύου [10].

- Εικονοποίηση υποστηριζόμενη από το υλικό Hardware-assisted virtualization: Αποτελεί την περίπτωση στην οποία ο guest είναι συμβατικός, χωρίς παραεικονοποιημένα μέρη, όμως το υλικό μπορεί να επιταχύνει τη διαδικασία μέσα από επεκτάσεις υλικού. Σε αυτές ανήκει και η VT-x που προαναφέρθηκε [11].

- Μικτή εικονοποίηση (Hybrid virtualization): Η προσέγγιση αυτή, η οποία αποτελεί και την κυρίαρχη τάση στα μοντέρνα συστήματα, συνδυάζει τις δύο παραπάνω βελτιστοποιήσεις. Έχει υποστήριξη και από το υλικό, αλλά και από το λογισμικό της εικονικής μηχανής, όσον αφορά την εικονοποίηση. Το τελικό αποτέλεσμα είναι πως στο σύστημα οι μόνες προνομιούχες εντολές οι οποίες εκτελούνται είναι αυτές τις οποίες μπορεί να διαχειριστεί αποδοτικά ο host στο υλικό, και ακόμα και αυτές γίνονται με ελεγχόμενο τρόπο μέσα από κλήσεις επόπτη (hypercalls), με το host να μπορεί με απλό τρόπο να παραδώσει τη ροή ελέγχου στην εικονική μηχανή.

Οι διαφορές μεταξύ όλων αυτών των ειδών εικονοποίησης είναι λεπτές, οπότε θα ήταν χρήσιμο να δούμε ένα παράδειγμα από τον πραγματικό κόσμο. Μία τέτοια ευκαιριά δίνει ο επόπτης Xen, ο οποίος προσφέρει πληθώρα εναλλακτικών όσον αφορά την εικονοποίηση. Αυτές διαφέρουν κυρίες σχετικά με το σύνολο των προνομιούχων ενεργειών τις οποίες πρέπει ο επόπτης να διαχειριστεί. Οι κατηγορίες στις οποίες μπορούν αδρομερώς να χωριστούν αυτές είναι οι παρακάτω:

- Συσκεύες (οι συνηθέστερες είναι δίσκοι και κάρτες δικτύου)

- Διακοπές (interrupts) και χρονιστές, οι οποίοι σχετιζονται άμεσα με το chipset του εκάστοτε μηχανήματος.

- Αρχικοποίηση Συστήματος (booting), η οποία μπορεί να είναι η παλαιότερη αρχικοποίηση σε 16-bit mode, ή η πιο μοντέρνα αρχικοποίηση με UEFI.

- Προνομιούχες εντολές, όπως αυτές οι οποίες δίνουν πρόσβαση σε καταχωρητές ελέγχου, με το σημαντικότερο από αυτούς να είναι ο καταχωρητής που δείχνει τη θέση στη μνήμη του καταλόγου σελίδων.

Τα διαφορετικά είδη αυτά διαφέρουν ως προς τον τρόπο με τον οποίο διαχειρίζονται τον έλεγχο των παραπάνω κλάσσεων ενεργειών και πόρων του guest. Απαριθμούνται παρακάτω, με τη σειρά με την οποία υιοθετήθηκαν [12][13]:

- Το PV mode καθίσταται εφικτό μόνο όταν ο guest συνεργάζεται με τον επόπτη, ώστε να αποφύγει την ανεξέλεγκτη εκτέλεση προνομιούχων εντολών. Αντ'αυτού, ο πρώτος εκτελεί δεδομένες ενέργειες όπως αυτές που αναφέρθηκαν παραπάνω για να σηματοδοτήσει στον επόπτη ένα δεδομένο αίτημα. Αυτός με τη σειρά του το ικανοποιεί, αν είναι έγκυρο, και ύστερα επιστρέφει τον έλεγχο στην εικονική μηχανή. Λόγω του σχεδιασμού αυτού, το λειτουργικό σύστημα του guest είναι απαραίτητο να ενθυλακώνει κάθε προνομιούχα εντολή μέσα σε μια πιο αφηρημένη πράξη, και να διαθέτει μια εναλλακτική, παραεικονοποιημένη υλοποίησή της. Καθώς, λοιπόν, απαιτούνται αλλαγές στο λειτουργικό της εικονικής μηχανής, με αυτό το είδος εικονοποίησης τρέχουν μόνο εικονικές μηχανές με λειτουργικά που διαθέτουν τέτοιου είδους λειτουργικότητα. Αφού όλες οι προνομιούχες ενέργειες είναι παραεικονοποιημένες, γεγονότα τα οποία θα ήταν πολύ γρήγορα 'πάνω στο μέταλλο', όπως διακοπές και προνομιούχες εντολές, έχουν χαμηλότερη απόδοση. Από την άλλη πλευρά, παραεικονοποιημένοι οδηγοί συσκευών έχουν παρεμφερή απόδοση με αυτή των συμβατικών συστημάτων. Η αρχικοποίηση συστήματος είναι επίσης απλούστερη, αφού ο guest έχει προσαρμοστεί για να ξεκινά πάνω από επόπτη [14].

- Το HVM Mode στο Xen αντιστοιχεί αποκλειστικά στην εικονοποίηση βοηθούμενη από υλικό, με την εικονοποίηση πλήρους συστήματος να είναι αδύνατη. Αυτό σημαίνει πως όλες οι προνομιούχες εντολές προσομοιώνονται, ευτυχώς με μειωμένο κόστος σε χρόνο χάρη στις επεκτάσεις του υλικού. Το είδος αυτό λειτουργεί μέσα από τη διατήρηση στον επεξεργαστή πληροφοριών σχετικά με το αν τρέχουν ως guests. Όταν εκτελούνται προνομιούχες εντολές στην εικονική μηχανή, ο επεξεργαστής επιστρέφει αποδοτικά στον επόπτη, χειρίζεται την εντολή, και επιστρέφει στην εικονική μηχανή. Αυτού του είδους η διαχείριση βοηθά με τη διαχείριση μνήμης, όπως έχει αναλυθεί παραπάνω, καθώς και με κάθε είδους προνομιούχες εντολές. Δε βοηθά, προφανώς, στην οδήγηση του εικονοποιημένου υλικού, το οποίο και πάλι προσομοιώνεται από το QEMU, ενώ και το chipset μαζί με τις σχετικές λειτουργίες, το οποίο είναι και αυτό ευθύνη του προσομοιωτή. Πρόσφατα, έχει προστεθεί στις επεκτάσεις υλικού υποστήριξη για την αποστολή διακοπών απέυθείας στην εικονική μηχανή αντί για τον host, με αποτέλεσμα την άρση του τελευταίου από τους παραπάνω περιορισμούς. Με τη χρήση, λοιπόν, επεκτάσεων όπως του ARM GICv3 [15] και μεταγενέστερων, δεν είναι απαραίτητη η έξοδος από την εικονική μηχανή για την αποστολή διακοπών, και το ζήτημα επιλύεται μια για πάντα.

- Το είδος παραεικονοποίησης HVM with PV drivers, κοινώς PV on HVM, αποτελεί μερικό συνδυασμό των δύο ειδών. Αυτή η ανάμειξη δε χρειάζεται ιδιαίτερη προσπάθεια για να επιτευχθεί - απλά χρησιμοποιείται πλήρης εικονοποίηση, ενώ εσωτερικά στον guest χρησιμοποιούνται παραεικονοποιημένοι οδηγοί συσκευών. Αξίζει να σημειωθεί πως κανένα άλλο γνώρισμα του PV mode δεν χρησιμοποιείται - η παραεικονοποίηση των διακοπών χρονιστή, για παράδειγμα, δεν αποτελεί επιλογή.

Poor Performance
Scope for Improvement
Optimal Performance

PV = Paravirtualized
VS = Software Virtualized (QEMU)
VH = Hardware Virtualized
HA = Hardware Accelerated

x86 Shortcut	Mode	With	Disk and Network	Interrupts & Timers	Boot Path	Privileged Instructions, Page Tables	QEMU Used
HVM / Fully Virtualized	HVM		VS	VS[1]	VS	VH	Yes
HVM + PV drivers	HVM	PV Drivers Installed	PV	VS[1]	VS	VH	Yes
PVHVM	HVM	PVHVM Capable Guest	PV	PV[2]	VS	VH	Yes
PVH	PVH	PVH Capable Guest	PV	HA[3]	PV[4]	VH	No
PV	PV		PV	PV	PV[5]	PV	No
ARM							
N/A	N/A		PV	VH	PV[6]	VH	No

Σχήμα 1.4: Απόδοση των διαφόρων μερών ενός συστήματος κατά τη χρήση των διαφόρων ειδών εικονοποίησης στο Xen. "XenModes" is licensed under CC SA 3.0

.

- Το είδος PVHVM αποτελεί βελτίωση από το παραπάνω ακριβώς στον τομέα αυτό. Το πλεονέκτημά του, λοιπόν, είναι η αποφυγή της χρονοβόρας προσομοίωσης διακοπών.

- Το είδος PVH, τέλος, είναι η τελευταία (και πιο επιτυχημένη) προσπάθεια να δημιουργηθεί ένα νέο είδος παραεικονοποίησης από τα δύο βασικά, PV και HVM, το οποίο να έχει τα πλεονεκτήματα και των δύο, και άρα τα μειονεκτήματα κανενός. Η τρέχουσα υλοποίηση αποκαλείται PVHv2, καθώς η πρώτη προσπάθεια δημιουργίας του συγκεκριμένου είδους κατέληξε σε αποτυχία. Ανάμεσα στα άλλα γνωρίσματά του, το είδος αυτό δε χρειάζεται καθόλου τον προσομοιωτή QEMU, αφού όλες ανεξαιρέτως οι προνομιούχες εντολές είτε έχουν αντικατασταθεί από παραεικονοποιημένες εκδόσεις, ή είναι εφικτό να γίνει η διαχείρισή τους αποκλειστικά από το υλικό.

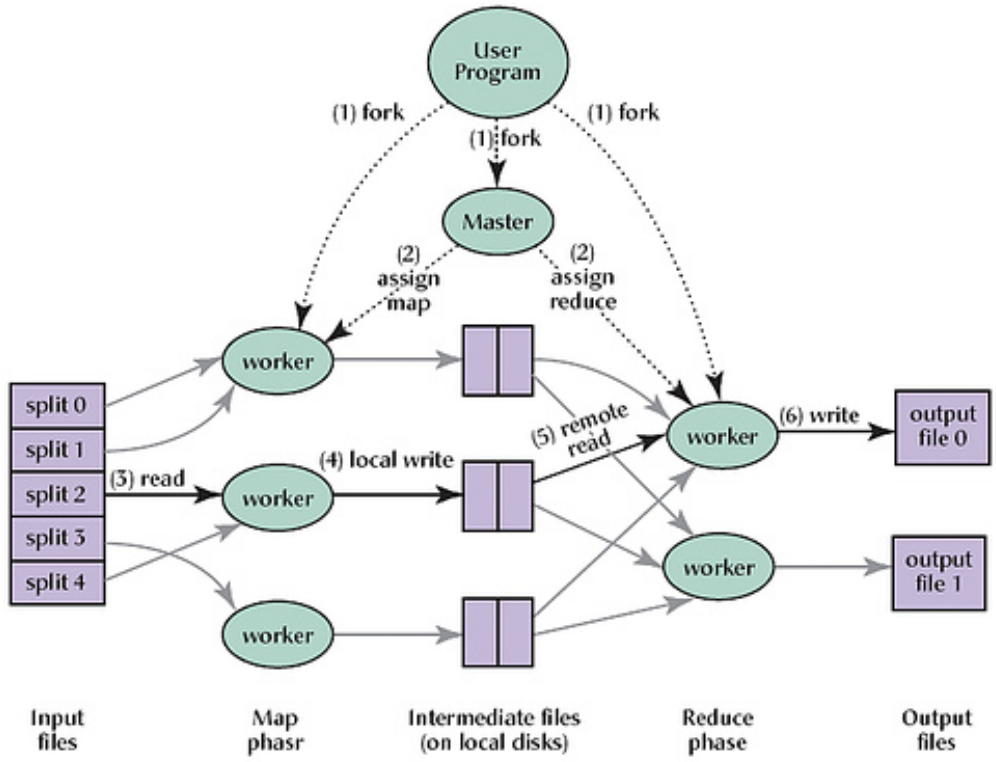
1.4 Υπολογιστικό Νέφος (Cloud Computing)

Όλες αυτές οι πρόοδοι στην εικονοποίηση συστημάτος έχει διαμορφώσει τις συνθήκες από τις οποίες προέκυψε η τάση του cloud computing: Το χαμηλό υπολογιστικό

κόστος της εικονοποίησης επιτρέπει τη χρήση εικονικών μηχανών αντί φυσικών μηχανημάτων με ελάχιστη επιπλέον κατανάλωση πόρων. Αποτελεί, λοιπόν, βέλτιστη λύση για την κατασκευή των υποδομών μιας εταιρείας η χρήση πόρων στο νέφος, οι οποίοι συναρμολογούνται σε εικονικές μηχανές. Οι πόροι αυτοί κοστίζουν λιγότερο από την αγορά τοπικών φυσικών εξυπηρετητών, καθώς οι πάροχοι νέφους (cloud providers) είναι σε θέση να εκμεταλλευτούν τη μείωση του κόστους ανά μονάδα των πόρων αυτών (economy of scale) και να προσφέρουν τέτοιου είδους υπηρεσίες σε εξαιρετικά ανταγωνιστικές τιμές. Η τάση αυτή της κατασκευής υπολογιστικών δομών στο νέφος μισθώνοντας παρόχους νέφους ονομάζεται Infrastructure as a Service (IaaS). Τα υπολογιστικά κέντρα αυτών των παρόχων έχουν το επιπλέον πλεονέκτημα πως μπορούν να εκμεταλλεύονται τη συνολική συμπεριφορά ως προς τη χρήση πόρων όλων των πελατών, έτσι ώστε να βελτιστοποιούν την απόδοσή τους, αλλά και την κατανάλωση ισχύος της υποδομής τους. Έτσι, βρίσκονται σε πλεονεκτική θέση σε σχέση με τις 'φάρμες' εξυπηρετητών μικρού και μεσαίου μεγέθους.

Για παράδειγμα, είναι δυνατή η απεικόνιση πολλαπλών εικονικών μηχανών οι οποίες ανήκουν στον ίδιο πελάτη στο ίδιο φυσικό μηχάνημα, με αποτέλεσμα η επικοινωνία μεταξύ τους να είναι εξαιρετικά αποδοτική. Άλλη μία επιλογή είναι η πολύπλεξη της κατανομής των εικονικών μηχανών μέσα στο νέφος με τέτοιο τρόπο ώστε στον ίδιο εξυπηρετητή να βρίσκονται εικονικές μηχανές με διαφορετικές ανάγκες. Ως αποτέλεσμα, έξυπνοι αλγόριθμοι κατανομής και χρονοπρογραμματισμού μπορούν να μειώσουν αισθητά τη μέγιστη κατανάλωση σε μνήμη, κύκλους επεξεργαστή, και bandwidth δικτύου, καθώς και να μεγιστοποιήσουν την εκμετάλλευση (utilization) του δικτύου. Επακόλουθο της μείωσης της μέγιστης κατανάλωσης στο σύστημα είναι πως τώρα η θεωρητική από την πραγματική ελάχιστη κατανάλωση διαφέρουν σημαντικά - αρκετά σημαντικά ώστε το κενό αυτό να μπορεί να χρησιμοποιηθεί από επιπλέον εικονικές μηχανές, για τις οποίες το νέφος θεωρητικά δεν έχει χωρητικότητα (overprovisioning). Πέραν των πλεονεκτημάτων για τον κάτοχο του νέφους, υπάρχει και η δυνατότητα στο χρήστη να αυξομειώνει δυναμικά τους πόρους τους οποίους έχει μισθώσει, έτσι ώστε να καταναλώνει ακριβώς όσους έχει ανάγκη ανά πάσα στιγμή. Το δυναμικό scale out αυτό αποτελεί στρατηγική η οποία είναι αδύνατη αν οι υπολογιστικοί πόροι βρίσκονται σε τοπικούς εξυπηρετητές.

Η διαθεσιμότητα τέτοιων ανήκουστων μέχρι πρότινως μεγεθών υπολογιστικής ισχύος, σε συνάρτηση με το όλο και πιο εύκολο 'στήσιμο' υποδομής στο νέφος έχει επιτρέψει την καθιέρωση νέων πρακτικών ως προς την αρχιτεκτονική λογισμικού. Μία από αυτές είναι οι μικροϋπηρεσίες (microservices), όπου μία διαδικτυακή υπηρεσία διασπάται σε πολλές ανεξάρτητες μικροϋπηρεσίες, κάθε μία από τις οποίες εκτελείται στο δικό της περιβάλλον. Αυτές τότε τοποθετούνται στο νέφος, με κάθεμία να παρέχεται τόσες φορές όσες χρειάζεται για να ικανοποιηθούν οι ανάγκες των χρηστών. Ο πληθυσμός κάθε μικροϋπηρεσίας μπορεί να προσαρμοστεί επί τόπου, και ο χρόνος ζωής κάθε μίας είναι ανεξάρτητος αυτών των άλλων. Η αρχιτεκτονική αυτή έχει όλα τα γνωστά πλεονεκτήματα μίας καλοσχεδιασμένης κατανεμημένης εφαρμογής: Κλι-



Σχήμα 1.5: Το μοντέλο προγραμματισμού MapReduce. "MapReduce Overview" is licensed under CC SA 3.0

.

μακώνει εξαιρετικά, μπορεί να συνεχίσει να εκτελείται ακόμα και εάν κάποια μέρη της παρουσιάσουν προβλήματα, και είναι ευκολότερη στη διαχείριση από μία μονολιθική εφαρμογή.

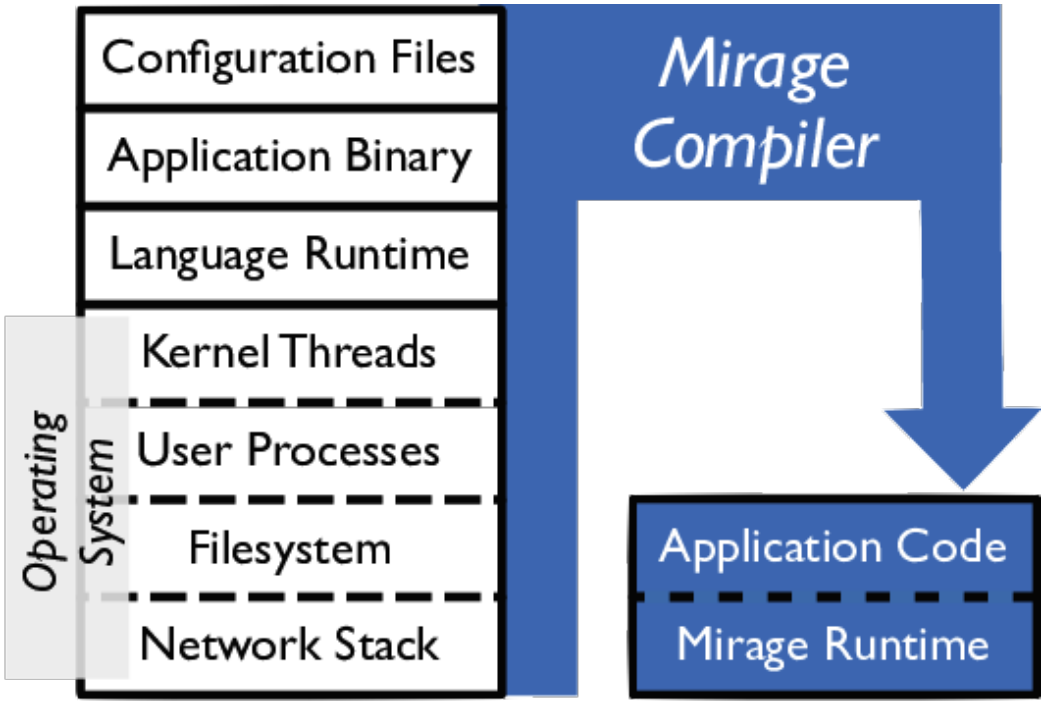
Άλλο ένα παράδειγμα μοντέρνας αρχιτεκτονικής λογισμικού η οποία έχει βρει μεγάλη επιτυχία χάρη στην ύπαρξη μεγάλης κλίμακας υπολογιστικών κέντρων είναι η MapReduce. Εφαρμογές που ακολουθούν αυτήν την τακτικεί μπορούν εύκολα να κλιμακώσουν 'προς τα έξω' (scale out), και αποτελούν σημαντικό μέρος των εφαρμογών που ασχολούνται με Big Data. Το πρώτο στάδιο στην αρχιτεκτονική είναι ο διαμερισμός των δεδομένων σε κομμάτια, τα οποία αποστέλλονται σε διαφορετικά μηχανήματα. Κάθε ένα από αυτά επεξεργάζεται την είσοδό του ανεξάρτητα των άλλων, επιτυγχάνοντας απόλυτα την παραλληλοποίηση του προβλήματος. Η έξοδος για κάθε ένα από τα κομμάτια αποστέλλεται σε μία συγκεντρωτική συνάρτηση, της οποίας το αποτέλεσμα είναι και το τελικό. Χαρακτηριστική εφαρμογή της μεθόδου αυτή είναι η μηχανή αναζήτησης Google [16].

Μία τρίτη καινοτομία η οποία προέκυψε από την εδραίωση της πρακτικής της χρήσης εικονικών μηχανών είναι η εικονοποίηση συναρτήσεων δικτύου (Network Function Virtualization, NFV). Η κεντρική ιδέα της πρακτικής αυτής είναι η αντικατάσταση εξειδικευμένου υλικού δικτύου, όπως τειχών προστατίας (firewalls) και δρομολογητών από εικονοποιημένα εξαρτήματα, τα οποία αποκαλούνται εικονοποιημένες συναρτήσεις δικτύου (Virtualized Network Functions (VNFs)). Αυτά τότε τοποθετούνται στο νέφος, και λειτουργούν από εκεί. Το πλεονέκτημα αυτής της προσέγγισης είναι, όπως και στις άλλες περιπτώσεις, η απλοποίηση της διαχείρισης του δικτύου, καθώς και η ευελιξία αυτού, αφού ο διαχειριστής μπορεί επιτόπου να προσθέσει ή να αφαιρέσει εικονικές μηχανές, ανάλογα με την κατάσταση.

1.5 Σχεδιαστικές τάσεις σχετικά με τον υπολογισμό νέφους

Ως αποτέλεσμα της εδραίωσης του υπολογισμού νέφους, έχουν βρεθεί νέα είδη λογισμικού συστήματος, βελτιστοποιημένα για εκτέλεση στο νέφος. Ένα από αυτά είναι το λογισμικό για containerization, το οποίο αναφέρθηκε σύντομα κατά τον ορισμό της έννοιας της εικονοποίησης. Το containerization είναι πιο συγκεκριμένα η εκτέλεση μίας εφαρμογής σε ένα απομονωμένο περιβάλλον, παρέχοντάς της ταυτόχρονα ένα σύνολο πόρων (πόρτες δικτύου, σύστημα αρχείων) διαφορετικά από αυτά τα οποία κατέχει το λειτουργικό σύστημα. Τα δεύτερα προβάλλονται στο χώρο ονομάτων της εφαρμογής, και παρουσιάζονται στην εφαρμογή με άλλη ονοματοδοσία. Για παράδειγμα, μια πόρτα πάνω από τη 1024 μπορεί να προβάλλεται στην εφαρμογή ως πόρτα 80 μεταθέτοντας ουσιαστικά την επιλογή της πόρτας την οποία θα χρησιμοποιήσει η εφαρμογή από την ίδια στο λογισμικό containerization. Έτσι πολυπλέκεται η εκτέλεση εφαρμογών οι οποίες ανταγωνίζονται για ένα συγκεκριμένο πόρο, και οι οποίες δε θα μπορούσαν να τρέξουν ως έχει σε έναν παραδοσιακό εξυπηρετητή. Το πιο δημοφιλές λογισμικό containerization είναι το Docker [4].

Επιστρέφοντας στην εικονοποίηση επιπέδου συστήματος στα πλαίσια του υπολογισμού νέφους, μία από τις πιο σύγχρονες καινοτομίες είναι η κατασκευή μονοπυρήνων (unikernels). Αυτοί είναι πυρήνες λειτουργικών συστημάτων οι οποίοι δεν είναι σχεδιασμένοι να εκτελούνται απέυθείας πάνω στο υλικό, αφού δεν έχουν τον απαραίτητο κώδικα για να καταφέρουν κάτι τέτοιο. Αντιθέτως, χρησιμοποιούν πάντα παραεικονοποίηση, με την επικοινωνία τους με το host να γίνεται μέσω κλήσεων επόπτη. Επιπλέον, είναι από την αρχή σχεδιασμένα να είναι πάρα πολύ στενά ενσωματωμένα με τις εφαρμογές τις οποίες εκτελούν, καθώς και οποιοδήποτε άλλο πρόγραμμα αυτές χρειάζονται. Για την ακρίβεια, όλα αυτά τα στοιχεία αποτελούν συχνά ένα ενιαίο εκτελέσιμο, καθώς μεταγλωττίζονται μαζί και ταυτόχρονα. Ένα ενδιαφέρον γνώρισμα των μονοπυρήνων είναι πως, αφού δεν έχουν την υποχρέωση να χειρίζονται απέυθείας υλικό, μπορούν να γραφούν σε οποιαδήποτε γλώσσα προγραμματισμού επιθυμεί ο προγραμματιστής, ανεξαρτήτως του πόσο αφαιρετική ή υψηλού επιπέδου είναι. Χαρα-



Σχήμα 1.6: Η στοίβα λογισμικού του MirageOS. "Unikernel Mirage Example" is licensed under CC SA 3.0

.

κτηριστικό γεγονός αποτελεί πως ο πιο δημοφιλής μονοπυρήνας είναι το Mirage OS [17], το οποίο είναι γραμμένο στη γλώσσα προγραμματισμού OCaml - μία γλώσσα η οποία δε θεωρείται καθόλου κατάλληλη για προγραμματισμό επιπέδου σύστηματος.

Μία ακόμα ενδιαφέρουσα εξέλιξη στο χώρο αυτό είναι τα anykernels. Αποτελούν το αποτέλεσμα μίας διαδικασίας κατά την οποία ένας συνηθισμένος πυρήνας χωρίζεται σε καλώς ορισμένα υποσυστήματα, σχεδόν ανεξάρτητα μεταξύ τους. Αυτά τότε αναδιατάσσονται σε αυθαίρετες διαρυθμίσεις, συνδυάζονται με την επιθυμητή εφαρμογή, και συνενώνονται σε ένα εκτελέσιμο το οποίο βρίσκεται όλο σε έναν ενιαίο χώρο διευθύνσεων. Το εκτελέσιμο αυτό μπορεί να τρέξει όπως και μία συμβατική εικονική μηχανή. Το τελικό αποτέλεσμα εμπεριέχει την εφαρμογή, μαζί με μόνο τα απαραίτητα κομμάτια του αρχικού πυρήνα τα οποία χρειάζονται για αυτήν, και αποκαλείται rump kernel. Αυτά βασίζονται σε πάρα πολύ μεγάλο βαθμό στο host, καλώντας τον ακόμα και για τις πιο βασικές λειτουργίες, όπως είναι η διαχείριση μνήμης και η δημιουγία διεργασιών. Η επικοινωνία με αυτόν επιτυγχάνεται, όπως συνηθίζεται κατά την παραεικονοποίηση, στις κλήσεις επόπτη.

Το τελικό συμπέρασμα από όλα αυτά τα παραδείγματα είναι πως υπάρχει μία ευρύτατη γκάμα μηχανισμών με τους οποίους ο guest μπορεί να επικοινωνήσει με το host, και πολλές και διάφορες αρχιτεκτονικές εικονοποίησης. Οι μέθοδοι οι οποίες μπορούν να υιοθετηθούν μπορούν να είναι από κλήσεις επόπτη έως προσομοίωση. Σε αυτήν την εργασία, η οποία έχει ως κεντρικό θέμα τη διαχείριση μνήμης, ένα αποτέλεσμα που μας ενδιαφέρει είναι πως η διαχείριση μνήμης μπορεί να γίνεται σε οποιαδήποτε επίπεδο της στοίβας λογισμικού, από τον ίδιο τον επεξεργαστή όταν πρόκειται για διάβασμα του καταλόγου σελίδων κατά την εικονοποίηση με βοήθεια υλικού, έως κλήσεις επόπτη για υψηλού επιπέδου αναθέσεις μνήμης όταν πρόκειται για rump kernels. Είναι λογικό, λοιπόν, να αναρωτηθούμε αν μπορούμε να βελτιώσουμε την απόδοση ενός εικονοποιημένου συστήματος μεταφέροντας την ευθύνη για τη διαχείριση της μνήμης σε διαφορετικά μέρη αυτού.

1.6 Διαχείριση μνήμης στο νέφος

Όπως αναφέρθηκε, ο υπολογισμός νέφους είναι αναμφισβήτητα μία από τις θεμελιώδεις τάσεις στη αρχιτεκτονική υπολογιστικών συστημάτων, με εικονικές μηχανές να αποτελούν τα βασικά συστατικά από τα οποία αποτελούνται μεγαλύτερες και πιο σύνθετες υπηρεσίες. Παρ΄όλα αυτά, επειδή οι εικονικές μηχανές αυτές τρέχουν συμβατικά λειτουργικά συστήματα τα οποια είναι σχεδιασμένα για να τρέχουν απέυθείας πάνω στο υλικό, το σύστημα στο σύνολό του υφίσταται ένα πρόβλημα παρεμφερές αυτού του κατακερματισμού (fragmentation) των πόρων τους στα συνηθισμένα μηχανήματα. Πιο συγκεκριμένα, επειδή κάθε guest έχει το δικό του εσωτερικό μηχανισμό για να διαχειρίζεται τη μνήμη του, χωρίς συναίσθηση του περιβάλλονοτας μέσα στο οποίο εκτελείται, όλες οι εικονικές μηχανές προσπαθούν ταυτόχρονα να βελτιστοποιήσουν τη διαχείριση της μνήμης τους με τον ίδιο τρόπο όπως και αν έτρεχαν μόνες τους σε ένα φυσικό μηχάνημα. Επιπλέον, οι hosts πολλές φορές συμπεριφέρονται σε αυτές όπως και στις συνηθισμένες διεργασίες, με αποτέλεσμα να μην λαμβάνουν υπ΄όψιν τη διαχείριση μνήμης την οποία αυτές εκτελούν. Ως αποτέλεσμα, και τα δύο μέρη αυτά του συστήματος διαχειρίζονται τη μνήμη έχοντας ελλειπείς πληροφορίες σχετικά με τη διαχείριση του συστήματος, και η διαχείριση αυτή είναι χαμηλής ποιότητας, επηρεάζοντας την απόδοση.

Το πρόβλημα της ανάθεσης μνήμης σε ένα σύνολο εικονικών μηχανών οι οποίες εκτελούνται στον ίδιο host, ανεξαρτήτως του είδους υπολογιστικού συστήματος στο οποίο τρέχουν (ενσωματωμένα μηχανήματα, επιτραπέζιοι ή φορητοί υπολογιστές, νέφος) λύνεται συνήθως με την εφαρμογή μίας εκ των τριών παρακάτω προσεγγίσεων, ή με ένα συνδυασμό αυτών:

- Υπερανάθεση πόρων (Overprovisioning): Σε κάθε εικονική μηχανή ανατίθονται αρκετοί πόροι ώστε να εξαλείφεται η πιθανότητα αυτή να χρειαστεί επιπλέον μνήμη. Αν και αυτή η προσέγγιση εξασφαλίζει την υψηλή απόδοση κάθε μηχανήματος, οδηγεί σε εξαιρετικά χαμηλή χρησιμοποίηση (utilization) πόρων, αφού

κάθε guest έχει αρκετή μνήμη για το χειρότερο πιθανό σενάριο, και συχνά δε χρησιμοποιεί μέρος της.

- Εναλλαγή στο δίσκο: Οι εικονικές μηχανές οι οποίες δεν έχουν άλλη μνήμη διαθέσιμη εναλλάσσουν σελίδες τους στο δίσκο. Αυτή η προσέγγιση, αν και αποδεκτή για δεδομένες εφαρμογές, μπορεί να οδηγήσει σε τεράστια μείωση της απόδοσης άλλων.

- Ballooning: Η πιο δημοφιλής των τριών προσεγγίσεων, επιτρέπει τη δυναμική αυξομείωση του μεγέθους της διαθέσιμης στο guest μνήμης. Αναλύεται αμέσως παρακάτω.

1.6.1 Ballooning

Η χρήση παραεικονοποιημένων συσκευών για την επίτευξη ως ένα βαθμό της συνεργασίας μεταξύ ενός guest και ενός host αποτελεί μία από τις πιο συνήθεις εφαρμογές της παραεικονοποίησης. Όσον αφορά τη διαχείριση μνήμης, μία τέτοια συσκευή είναι το memory balloon, το οποίο επιτρέπει την ενοποίηση της διαχείρισης και των δύο αυτών στοιχείων του συστήματος μέσα από τη δυνατότητα μεταφοράς της κυριότητας μέρους του φυσικού χώρου διευθύνσεων του guest στο host. Η κατοχή μνήμης από το χώρο αυτό γίνεται αντιληπτή με διαφορετικό τρόπο για καθέναν από τους δύο:

- Από την πλευρά του guest, η μνήμη την οποία αυτός έχει είναι απλά μνήμη - φυσικές σελίδες οι οποίες κυκλοφορούν μεσα στο υποσύστημα διαχείρισης μνήμης του. Το ίδιο ισχύει και για το host, για τη μνήμη την οποία αυτός κατέχει.

- Από την πλευρά του host, όμως, η μνήμη την οποία έχει ο guest είναι, όπως συμβαίνει και με τις υπόλοιπες διεργασίες, εικονική μνήμη. Αυτό σημαίνει πως μπορεί να αντιστοιχιστεί σε φυσική μνήμη κατά βούληση.

Από την πλευρά του guest, λοιπόν, μέρος της μνήμης του μπορεί να δοθεί στο host αναθέτοντάς την στη συσκευή balloon. Η χρησιμότητα αυτής της συσκευής έγκειται στο ότι οι σελίδες οι οποίες της δίνονται δε χρησιμοποιούνται ποτέ, οπότε ο host μπορεί να σταματήσει να τις αντιστοιχεί σε φυσικές σελίδες του, μειώνοντας στην ουσία το μέγεθος της πραγματικής μνήμης που μπορεί να χρησιμοποιήσει η εικονική μηχανή. Όταν ο guest παρουσιάζει έλλειψη μνήμης, ανακτά τις σελίδες από τη συσκευή, και η δική του φυσική μνήμη επανέρχεται στην αρχική κατάστασή της.

Συνολικά, το ballooning ως λύση δυστυχώς παρουσιάζει πληθώρα προβλημάτων, διότι τα λειτουργικά συστήματα είναι κατ'εξοχίν σχεδιασμένα να χρησιμοποιούν όση περισσότερη μνήμη μπορούν. Αυτό είναι απόρροια του γεγονότος πως προορίζονται να εκτελεστούν πάνω σε πραγματικό υλικό, και το να μη χρησιμοποιούν όλη τη φυσική μνήμη σε αυτήν την περίπτωση ισοδυναμεί με τη μείωση του utilization. Ως αποτέλεσμα, ακόμα και στις περιπτώσεις όπου η συσκευή μεγεθύνει το ποσό μνήμης που

'απορροφά' αυτόκλιτα (self-ballooning) χωρίς την παρέμβαση της εικονικής μηχανής ή του host, ο τελευταίος είναι παγιδευμένος με τον guest σε μία διελκυστίνδα για την κατοχή των φυσικών σελίδων του. Ένα από τα πιο χαρακτηριστικά παραδείγματα αυτού του φαινομένου είναι η κρυφή μνήμη σελίδων (page cache) του λειτουργικού συστήματος Linux, η οποία είναι και μείζων καταναλωτής των αχρησιμοποίητων σελίδων του συστήματος, και η οποία αποσκοπεί στη μείωση της κίνησης από και προς το δίσκο. Η κρυφή μνήμη αυτή προσπαθεί κάθε φορά που διαβάζεται κάτι ή γράφεται κάτι στο δίσκο να αντικαταστήσει την ενέργεια αυτή με ένα απλό διάβασμα μνήμης, ενώ μειώνεται σε μέγεθος όταν το σύστημα βρίσκεται υπό πίεση από πλευράς κατανάλωσης μνήμης. Ο σχεδιασμός αυτός είναι ιδανικός για συμβατικά συστήματα, αφού έτσι χρησιμοποιούνται πλήρως όλοι οι διαθέσιμοι πόροι. Σε ένα περιβάλλον όπου χρησιμοποιείται εικονοποίηση, όμως, αυτό δεν ισχύει, καθώς η μεγέθυνση της page cache συνεπάγεται τη στέρηση από το host της μνήμης την οποία θα έπαιρνε μέσα από τη λειτουργία του balloon driver. Το τελικό αποτέλεσμα είναι η κρυφή μνήμη αυτή να συμβάλλει στην πίεση μνήμης (memory pressure) την οποία δέχεται το σύστημα, αντί να την αντιμετωπίζει [18].

Ένα εναλλακτικό μοντέλο είναι να δίνεται στο guest όσο το δυνατόν λιγότερη φυσική μνήμη (φυσική από τη δική του οπτική γωνία), μαζί με τη δυνατότητα να διαχειρίζεται ένα επιπλέον ποσό μνήμης αποκλειστικά μέσω κλήσεων επόπτη. Κατ'αυτόν τον τρόπο, το υποσύστημα διαχείρισης μνήμης του guest έχει λιγότερες αρμοδιότητες, και ασκεί λιγότερη επιρροή στην διαχείριση μνήμης του συνολικού συστήματος, με το host να λαμβάνει κεντρικά τις περισσότερες αποφάσεις. Αυτή η κατάσταση επιτρέπει στον τελευταίο να εφαρμόσει πιο αποδοτικές πολιτικές διαχείρισης μνήμης.

1.7 Ο μηχανισμός Transcendent Memory

Ο μηχανισμός Transcendent Memory (tmem) αποτελεί μία προσέγγιση στο πρόβλημα της συνεργατικής διαχείρισης μνήμης μεταξύ guest και host η οποία σχεδιάστηκε από την εταιρεία Oracle κατά το 2009, και η οποία ενσωματώθηκε στον κώδικα του πυρήνα του Linux σύντομα μετά τη δημοσίευσή του. Η κεντρική ιδέα ήταν πως, αφού οι εικονικές μηχανές είναι ικανές να εκμεταλλευτούν τις παραεικονοποιημένες συσκευές, είναι εφικτό να δημιουργηθεί μία η οποία να λειτουργεί ως αποθήκη κλειδιού-τιμής (key-value store) για όσες σελίδες τους δε χρειάζονται την εκάστοτε στιγμή. Παραδείγματα τέτοιων σελίδων αποτελούν οι σελίδες οι οποίες έχουν αποσταλεί από το λειτουργικό στο χώρο εναλλαγής του δίσκου - αυτές αποθηκεύονται στη βάση αυτή πολύ πιο γρήγορα από ότι θα αποστέλονταν σε μία συσκευή μπλοκ. Αυτό το σύνολο μνήμης (memory pool) είναι τελείως αδιαφανές στην εικονική μηχανή - αφού οι σελίδες που είναι εκεί δεν έχουν διεύθυνση, είναι αδύνατον ο guest να αναφερθεί σε αυτές με εντολή load ή store. Αφού, λοιπόν, τα περιεχόμενα της βάσης είναι μη προσβάσιμα, ο host μπορεί να τα μετασχηματίσει και να τα αποθηκεύσει σε οποιαδήποτε μορφή επιθυμεί. Μόνος περιορισμός είναι να μπορεί να τα επαναφέρει στην αρχική τους κατάσταση όταν του ζητηθεί από την εικονική μηχανή μέσω μίας κλήσης επόπτη.

Το tmem όχι μόνο μπορεί να χρησιμοποιηθεί ως πλήρης λύση ως έχει, αλλά μπορεί και να εμπλουτιστεί με την προσθήκη στο πίσω μέρος (backend) του μεθόδων οι οποίες εφαρμόζονται στα δεδομένα, οι οποίες δεν ήταν εφικτές πριν. Μία τέτοια λύση αποτελεί για παράδειγμα το zswap module: Οι σελίδες οι οποίες αποθηκεύονται από τις μηχανές συμπιέζονται, έτσι ώστε να καταλαμβάνουν λιγότερο χώρο όταν δε χρησιμοποιούνται. Άλλη μία δυνατή επέκταση είναι η μεταφορά των δεδομένων σε διαφορετικό φυσικό μηχάνημα, και η επαναφορά τους στο αρχικό μόνο όταν αυτά ζητηθούν από την εικονική μηχανή. Ένα τέτοιο σύστημα επιτρέπει νέες αρχιτεκτονικές υλικού στα υπολογιστικά κέντρα, με εξυπηρετητές οι οποίοι είναι εξειδικευμένοι σε ένα δεδομένο υπολογιστικό πόρο. Έτσι, μπορεί να υπάρχουν μηχανήματα με πολύ δυνατούς επεξεργαστές, και μηχανήματα με μεγάλα ποσά μνήμης, τα οποία συνεργάζονται για την εκτέλεση των εικονικών μηχανών, οι οποίες εκμεταλλεύονται τους πόρους και των δύο.

Το tmem είναι τελείως ανεξάρτητο από το υπόλοιπο του συστήματος διαχείρισης μνήμης ως προς τη λειτουργικότητά του, και προστίθεται σε αυτό πολύ εύκολα. Για παράδειγμα, στον πυρήνα του Linux η ενσωμάτωση του μηχανισμού αποτελείται απλά από ένα μικρό σύνολο ελέγχων στο swap subsystem. Στους χρήστες του μηχανισμού tmem (frontends) συμπεριλαμβάνονται το frontswap και το cleancache, καθένα από τα οποία χρησιμοποιούν διαφορετικού είδους σύνολα μνήμης tmem:

- Το frontswap χρησιμοποιεί μόνιμα σύνολα (persistent pools). Αυτό σημαίνει πως τα δεδομένα τα οποία αποθηκεύονται σε αυτά διατηρούνται πάντα. Ως αποτέλεσμα, μια επιτυχημένη αποθήκευση συνεπάγεται τη σίγουρη επιτυχία της ανάσυρσης των δεδομένων από το σύνολο.
- Το cleancache χρησιμοποιεί παροδικά σύνολα (ephemeral pools). Τα σύνολα αυτά λειτουργούν όπως και πολλές κρυφές μνήμες, αποβάλλοντας δεδομένα δίχως προειδοποίηση για να δημιουργήσουν χώρο για νέες προσθήκες. Έτσι, η ανάσυρση δεδομένων αποθηκευμένων στα σύνολα αυτά μπορεί να αποτύχει. Προφανώς, τα σύνολα αυτά είναι χρήσιμα μόνο για την αποθήκευση δεδομένων τα οποία έχουν κάποιο αντίγραφο κάπου αλλού, από όπου θα μπορέσουν να βρεθούν σε περίπτωση που διαγραφούν από το σύνολο.

Το frontswap χρησιμοποιείται ως κρυφή μνήμη για το χώρο αποθήκευσης, επιτρέποντας στο guest να στείλει σε αυτό σελίδες οι οποίες εναλλακτικά θα πήγαιναν στο χώρο εναλλαγής. Εφ'όσον η εικονική μηχανή δεν υπερβαίνει το μερίδιό της στο σύνολο μνήμης, αυτό σημαίνει πως μπορεί να χρησιμοποιεί ανά πάσα στιγμή δεδομένα μεγέθους μεγαλύτερου από αυτό της φυσικής μνήμης του, με κόστος μία μικρή μείωση της απόδοσης.

Το cleancache λειτουργεί ως "victim cache for clean pages", σύμφωνα με το δημιουργό της. Χρησιμοποιείται για διάβασμα δεδομένων, και χρησιμοποιείται για να μη ζητούνται από τη συσκευή μπλοκ τα ίδια δεδομένα ξανά και ξανά. Αφού οι σελίδες

υπάρχουν πάντα και στο δίσκο, το σύνολο μνήμης μπορεί να διαγράψει μέρος των δεδομένων του όποτε επιθυμεί.

1.8 Στόχοι και δομή της παρούσας εργασίας

Στόχος της παρούσας εργασίας είναι η εξέταση της δυνατότητας ανάπτυξης μηχανισμών παραεικονοποίησης με τελικό στόχο τη βελτίωση της απόδοσης των εφαρμογών οι οποίες εκτελούνται στο νέφος, επιτρέποντάς τους να συνεργάζονται άμεσα και εκτενώς με το host ως προς τη διαχείριση μνήμης. Το κείμενο είναι διαρυθμισμένο ως εξής: Στο κεφάλαιο 2 αναλύεται εκτενώς η αρχιτεκτονική του μηχανισμού tmem, και παρουσιάζεται επιπλεόν η δική μας συμβολή, η οποία ονομάζεται User Transcendent Memory (utmem), μαζί με τις σχεδιαστικές επιλογές μας και λεπτομέρειες σχετικές με την υλοποίησή της. Το κεφάλαιο 3 παρουσιάζει τα αποτελέσματα των μετρήσεών μας σχετικά με την απόδοση του μηχανισμού μας σε διάφορα περιβάλλοντα. Το κεφάλαιο 4 περιέχει αναφορές σε μηχανισμούς οι οποίοι απαντώνται στη βιβλιογραφία και σχετίζονται με το θέμα της διαχείρισης μνήμης το νέφος, καθώς και τα συμπεράσματά μας και μία περίληψη.

Κεφάλαιο 2

Τεχνικό Υπόβαθρο και Σχεδιαστικές Επιλογές

2.1 Τεχνικό Υπόβαθρο

Ο μηχανισμός tmem μπορεί να χρησιμοποιηθεί για να επιτευχθεί η συγκέντρωση και σύμπτυξη της μνήμης σε συστήματα τα οποία εκτελούν πολλαπλές εικονικές μηχανές. Πιο συγκεκριμένα, επιχειρεί να λύση το πρόβλημα της μεταβαλλόμενης ανάγκης για μνήμη των εικονικών μηχανών, πρόβλημα το οποίο προξενεί σημαντική μείωση της απόδοσης σε συστήματα τα οποία βασίζονται αποκλειστικά στο μηχανισμό ballooning. Η ιδέα για το μηχανισμό αυτόν προέκυψε από την εργασία των Magenheimer et al. [19].

Ο μηχανισμός tmem είναι μία αποθήκη κλειδιού-τιμής η οποία είναι προσβάσιμη μόνο άμεσα, μέσω της χρήσης κλήσεων επόπτη. Η αποθήκη αυτή συγκροτείται από μη χρησιμοποιούμενη μνήμη στο host η οποία συγκεντρώνεται σε ένα κοινό σύνολο, το οποίο διαμοιράζεται στις εικονικές μηχανές. Μία από τις θεμελιώδεις ιδιότητες του μηχανισμού είναι πως η αλληλεπίδραση με αυτό το σύνολο γίνεται σχεδόν αποκλειστικά μέσα από τα δύο αιτήματα: PUT και GET (υπάρχουν και άλλα, αλλά είναι βοηθητικά σε αυτά τα δύο). Έχοντας καλώς ορισμένα σημεία επικοινωνίας με τους χρήστες του, ο μηχανισμός αποσυμπλέκει την υλοποίηση των συνόλων δεδομένων από τη διεπαφή με τους χρήστες. Έτσι, τα δεδομένα στα σύνολα μπορούν να αναπαρίστανται με διάφορους τρόπους, όπως για παράδειγμα σε συμπιεσμένη μορφή [20]. Ακόμα και στην περίπτωση όπου τα δεδομένα αποθηκεύονται τοπικά και ως έχει, όπως στην περίπτωση της υλοποίησης του μηχανισμού στο επόπτη Xen [21], οι χρήστες και πάλι δεν μπορούν να έχουν πρόσβαση σε αυτά χωρίς να στείλουν σχετικό αίτημα στον επόπτη. Αυτό σημαίνει πως ο host διατηρεί τον έλεγχο της μνήμης του συστήματος ανά πάσα στιγμή.

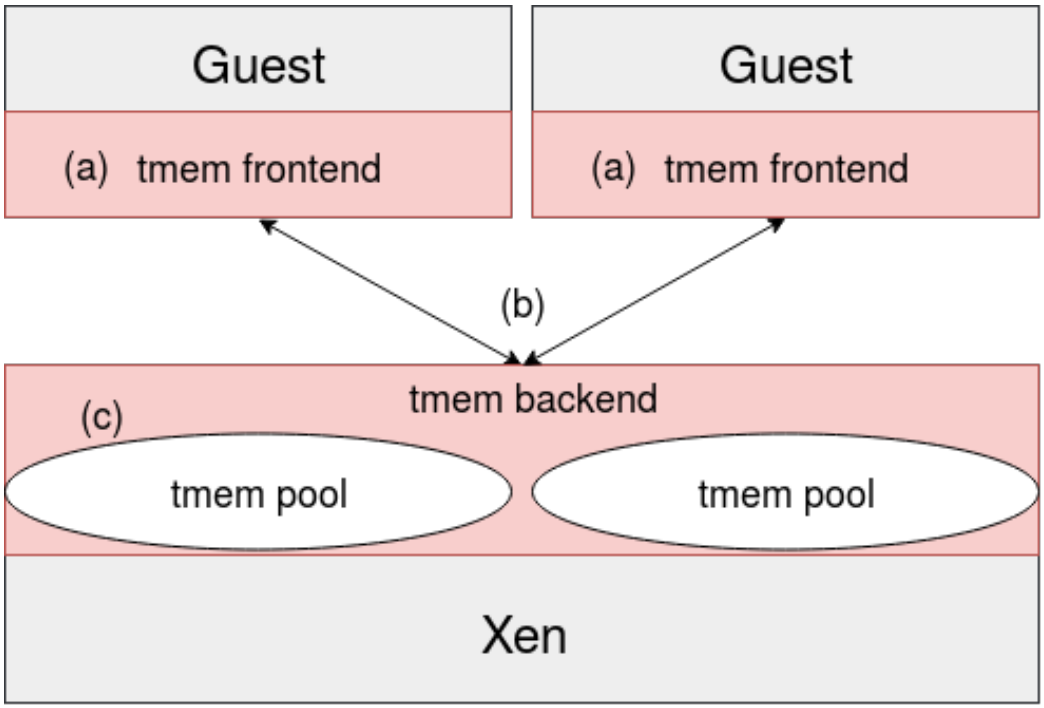
Το σχήμα 2.1 παρουσιάζει την τοπολογία του μηχανισμού, και πιο συγκεκριμένα την τοποθεσία στο σύστημα των συνόλων μνήμης tmem, καθώς και των ενδιάμεσων μνημών (buffers) του guest στην υλοποίηση για τον επόπτη Xen. Κάθε ένας από αυτούς έχει το δικό του χρήστη tmem (a), ο οποίος είναι και το μόνο κομμάτι του μηχανισμού το οποίο βρίσκεται στην εικονική μηχανή. Επικοινωνεί με το host μέσω κλήσεων επόπτη (b), οι οποίες επεξεργάζονται και ικανοποιούνται από αυτόν στο πίσω μέρος (c). Το τελευταίο αποθηκεύει ή ανασύρει τα δεδομένα από τα σύνολα μνήμης tmem τα οποία έχει στην κατοχή του.

Τα σύνολα μνήμης χαρακτηρίζονται ως ιδιωτικά ή διαμοιραζόμενα, ανάλογα με το αν τα δεδομένα που αποθηκεύονται εκεί από μία εικονική μηχανή είναι προσβάσιμα και σε άλλες, ή μόνο σε αυτήν η οποία τα έστειλε στο πίσω μέρος. Η αρχική περιγραφή του μηχανισμού tmem κάνει λόγο επίσης για παροδικά ή μόνιμα σύνολα μνήμης, ανάλογα με το αν τα δεδομένα που αποθηκεύονται σε αυτές διατηρούνται μόνιμα ή όχι. Επειδή τα παροδικά σύνολα μπορούν να διαγράψουν τα περιεχόμενά τους κατά βούληση για να απελευθερώσουν τη μνήμη την οποία χρησιμοποιούν, μπορούν να αποτελέσουν το βασικό συστατικό οντοτήτων όπως κρυφές μνήμες για σελίδες. Αντίθετα, τα μόνιμα σύνολα είναι βέβαιο πως διατηρούν τα αποθηκευμένα δεδομένα, οπότε και είναι πιο κατάλληλα ως μέρη μηχανισμών τα οποία προσφέρουν υψηλής ταχύτητας εναλλακτικές σε μέσα αποθήκευσης τα οποία κοστίζουν σε χρόνο. Σε αυτήν την εργασία επικεντρωνόμαστε στα μόνιμα σύνολα.

Χρήστες του μηχανισμού tmem είναι, όπως αναφέρθηκε, οι μηχανισμοί frontswap και cleancache. Αυτοί υλοποιούνται ως modules για Linux guests οι οποίοι τρέχουν πάνω από έναν επόπτη Xen. Αποτελούνται από "hooks" στο υποσύστημα μνήμης ή εισόδου-εξόδου, αντίστοιχα, τα οποία μπορούν να χρησιμοποιηθούν για να πραγματοποιήσουν αιτήματα tmem στον επόπτη, αντί να καταφύγουν σε πιο χρονοβόρες εναλλκτικές όπως η πρόσβαση στο δίσκο. Το cleancache λειτουργεί ως κρυφή μνήμη σελίδων για την εικονική μηχανή, την οποία όμως διαχειρίζεται ο host. Η λειτουργία της περιλαμβάνει απόπειρες να αποθηκευτούν μπλοκ τα οποία διαβάστηκαν από το δίσκο σε ένα παροδικό σύνολο μνήμης. Σε περίπτωση όπου το ίδιο μπλοκ επιχειρηθεί να διαβαστεί αργότερα, και το αντίγραφο στο σύνολο μνήμης δεν έχει διαγραφεί, ο μηχανισμός μπορεί να το ανασύρει και να αποφύγει ως αποτέλεσμα μία πρόσβαση στο δίσκο.

Όπως προαναφέρθηκε, υπάρχει και το frontswap, το οποίο αποσκοπεί στη μείωση του ρυθμού εναλλαγής στο δίσκο, και στη μείωση της κίνησης από και προς αυτόν γενικότερα [22]. Το σχήμα 2.2 σκιαγραφεί τη σχέση μεταξύ το υποσύστημα διαχείρισης μνήμης του guest, του frontswap, και του πίσω μέρους του μηχανισμού tmem. Πριν ο guest καταφύγει στην εναλλαγή στο δίσκο, στέλνει πρώτα ένα αίτημα στο frontswap (a) για να επιχειρήσει να σώσει τη σελίδα. Ο μηχανισμός επιχειρεί να το ικανοποιήσει, ανασύροντας την υλοποίηση της ενέργειας αυτής, η οποία είναι συνήθως κλήση επόπτη (b), και στέλνοντας ένα αίτημα PUT στο πίσω μέρος (c). Το πίσω μέρος

απαντά θετικά ή αρνητικά, ανάλογα με το αν η αποθήκευση πέτυχε, και το αποτέλεσμα περνά από όλα τα μέρη του μηχανισμού μέχρι να φτάσει στο υποσύστημα του πυρήνα το οποίο ξεκίνησε το αίτημα. Αν η απάντηση είναι θετική, τότε η σελίδα δε χρειάζεται να αποσταλεί στο δίσκο, και το κόστος μίας εγγραφής σε συσκευή μπλοκ αντικαταστάθηκε από μία κλήση επόπτη.



Σχήμα 2.1: Η τοποθεσία των μερών του μηχανισμού tmem στη υλοποίησή του στον επόπτη Xen

2.2 Λόγοι για την κατασκευή του μηχανισμού utmem

Όπως προαναφέρθηκε, ο μηχανισμός tmem έχει αποδειχθεί πως ενσωματώνεται πολύ καλά με το υποσύστημα μνήμης του guest, οπότε το σημείο αυτό του πυρήνα φαίνεται πως είναι ιδιαίτερα ατάλληλο για να προστεθεί εκεί ο μηχανισμός. Αν και αυτή η προσέγγιση λειτουργεί καλά, τα συστήματα με τη διαρύθμιση αυτή παρουσιάζουν το πρόβλημα το οποίο ο μηχανισμός υποτίθεται πως επιλύει, απλά σε άλλο σημείο: Οι εφαρμογές και πάλι δεν μπορούν να παράσχουν πληροφορίες σχετικά με τον τρόπο με τον οποίο χρησιμοποιούν μνήμη στο host, και αφήνουν αντ'αυτού το υποσύστημα διαχείρισης μνήμης της εικονικής μηχανής να λαμβάνει την πρωτοβουλία σχετικά με το ποια δεδομένα θα αποσταλούν στο πίσω μέρος του μηχανισμού. Η συμπεριφορά αυτή

είναι λογική όταν η εικονική μηχανή εκτελεί πολλαπλές διεργασίες, και άρα βγάζει νόημα το υποσύστημα μνήμης της να πολυπλέκει τα αιτήματα αυτών για μνήμη και να επικοινωνεί εκείνο με το μηχανισμό tmem. Σε εικονικές μηχανές, όμως, οι οποίες προορίζονται να τρέχουν αποκλειστικά μία εφαρμογή, η παρεμβολή του πυρήνα του guest αποτελεί πιο πολύ πρόβλημα από ότι βοήθεια.

Σε περιπτώσεις όπως αυτές, λοιπόν, βγάζει νόημα να επιτρέπουμε στα ίδια τα προγράμματα να λαμβάνουν άμεσα αποφάσεις και να δηλώνουν ρητά στο μηχανισμό ποια δεδομένα να αποστέλλονται στο host, αντί να αλληλεπιδρούν με αυτόν έμμεσα μέσα από σφάλματα σελίδας. Για να επιτευχθεί αυτό, η δυνατότητα στο χώρο χρήστη να εκτελεί αιτήματα tmem προσφέρεται μέσα από μία αφηρημένη συσκευή, παρακάμπτοντας τον πυρήνα της εικονικής μηχανής. Έτσι, αφαιρείται ακόμα ένα στρώμα από τη στοίβα λογισμικού η οποία λαμβάνει αποφάσεις για τη διαχειριση μνήμης στο σύστημα, και η διαδικασία διαχείρισης απλοποιείται ακόμα περισσότερο, εκμεταλλευόμενη το περιβάλλον στο οποίο τρέχει. Ο νέος μηχανισμός αυτός για την προσφορά υπηρεσιών tmem σε επίπεδο χώρου χρήστη ονομάζεται Userspace Transcendent Memory (utmem).

2.3 Σχεδιαστικές Επιλογές

Η χρήση του μηχανισμού tmem αποκλειστικά στον πυρήνα, και η παρουσίασή του ως έναν μηχανισμό ο οποίος δε γίνεται αντιληπτός από το υπόλοιπο σύστημα, αποτελούν αυθαίρετους περιορισμούς οι οποίοι μπορούν με ελάχιστο κόπο να αρθούν. Όχι μόνο αυτό, αλλά η άρση τους, αντί να μειώνει, αυξάνει την αποδοτικότητά του. Πιο συγκεκριμένα, έως τώρα τα αιτήματα tmem γίνονταν μόνο στα πλαίσια του υποσυστήματος μνήμης και εισόδου-εξόδου, με αποτέλεσμα κάθε κλήση να χρειάζεται να περάσει μέσα από τη στοίβα I/O του guest.

Αυτή η διαδικασία κλήσεων δεν είναι εγγενής στο μηχανισμό. Αντιθέτως, ο λόγος για τον οποίο ο μηχανισμός tmem υπερισχύει προσεγγίσεων όπως η χρήση ως συσκευή εναλλαγής δίσκων μνήμης τυχαίας προσπέλασης (ramdisks) είναι πως η αλληλεπίδραση με τα σύνολα μνήμης γίνονται χωρίς να πρέπει να χρησιμοποιηθεί μεγάλο κομμάτι της στοίβας I/O. Αντί αυτής χρησιμοποιείται μία συγκριτικά πολύ πιο γρήγορη κλήση επόπτη. Ένας από τους κύριους στόχους του utmem είναι να παρακάμψει εντελώς τη στοίβα I/O και να επικοινωνεί με τη μνήμη του host που της αναλογεί μόνο μέσα από κλήσεις επόπτη, με αποτέλεσμα ένα γρηγορότερο και απλούστερο σύστημα.

Ο κύριο στόχος κατά τη σχεδίαση του μηχανισμού utmem ήταν η εύκολη ενσωμάτωση σε εφαρμογές οι οποίες χειρίζονται τα δεδομένα τους μέσα από κλήσεις ενός σαφώς ορισμένου API. Ο μηχανισμός μας μπορεί να παρεμβληθεί, τότε, ανάμεσα στο τελευταίο και την εφαρμογή, αντικαθιστώντας τις κλήσεις εγγραφής και ανάγνωσης. Για να μπορεί να γίνει αυτό, η υλοποίηση του πίσω μέρους στο host είναι απαραίτητο

να υποστηρίζει την εξυπηρέτηση αιτημάτων για αυθαίρετου είδους και μεγέθους δεδομένα, όχι μόνο για σελίδες. Το σύνολο μνήμης πρέπει να είναι αδιαφανές και όχι άμεσα προσβάσιμο στις εικονικές μηχανές, να αποτελείται από σύνολα μνήμης tmem, και να είναι πλήρως διαχειρίσιμο χρησιμοποιώντας αποκλειστικά κλήσεις tmem.

Ο μηχανισμός utmem αποτελείται από τρία μέρη, τα οποία επικοινωνούν μεταξύ τους χρησιμοποιώντας κλήσεις tmem. Αυτά τα τρία μέρη είναι:

- Η συσκευή utmem, η οποία προσφέρει τη δυνατότητα εφαρμογής επιπλέον κανόνων στο μηχανισμό, και η οποία προσφέρει τη δυνατότητα αποστολής αιτημάτων tmem στις διεργασίες
- Ο χρήστης tmem, ο οποίος είναι υπεύθυνος για την επικοινωνία με το πίσω μέρος, και ο οποίος καθορίζει τον ακριβή τρόπο με τον οποίο επιτυγχάνεται αυτή (κλήσεις επόπτη, μηνύματα μέσω δικτύου, κλπ.) ανάλογα με το πίσω μέρος που χρησιμοποιείται
- Το πίσω μέρος, το οποίο περιέχει το σύνολο μνήμης tmem, είναι υπεύθυνο για την αποθήκευση και ανεύρεση δεδομένων, και ικανοποιεί αιτήματα

Όσον αφορά τις μη λειτουργικές απαιτήσεις του μηχανισμού, η υλοποίησή μας επιχειρεί να είναι εντελώς αρθρωτή, με τα τρία συστατικά της να είναι ανεξάρτητα το ένα του άλλου. Αυτό μας επιτρέπει να εφαρμόσουμε την υλοποίησή μας σε πληθώρα τόσο εφαρμογών χρήστη, όσο και λειτουργικών συστημάτων, με ελάχιστες μετατροπές. Επιπλέον, επιθυμούμε να μπορούμε να κατασκευάζουμε αυθαίρετου μεγέθους μηχανισμούς από χρήστες και πίσω μέρη utmem, προωθώντας αιτήματα από το ένα συστατικό στο άλλο. Έτσι, επιλέγοντας προσεκτικά τα συστατικά στοιχεία της στοίβας λογισμικού μπορούμε να δημιουργήσουμε αυθαίρετες ιεραρχίες μνήμης βασιζόμενες στο μηχανισμό tmem, με διαφορετικά επίπεδα να παρουσιάζουν διαφορετικές χωρητικότητες, υλοποιήσεις πίσω μέρους, και ταχυτήτων με τις οποίες εξυπηρετούνται τα αιτήματα.

Η παρούσα υλοποίηση είναι γενική και είναι κατάλληλη για τη χρήση κάθε είδους δεδομένων, από φυσικές σελίδες έως ζεύγη κλειδιού-τιμής. Τα συστήματα τα οποία τη χρησιμοποιούν αποδεικνύονται επίσης πως είναι πιο αποδοτικά τόσο από συμβατικά συστήματα, όσο και από συστήματα τα οποία κάνουν χρήση του μηχανισμού tmem μόνο στο υποσύστημα μνήμης τους. Αυτή η επιτάχυνση σε σχέση με τους υπάρχοντες μηχανισμούς είναι εφικτή χωρίς να εκμεταλλευόμαστε τυχόν ιδιαιτερότητες των δεδομένων της εφαρμογής. Ο μηχανισμός μπορεί να τροποποιηθεί με μικρές αλλαγές ώστε να λαμβάνει υπ'όψιν τη φύση της εφαρμογής με σκοπό την επίτευξη ακόμα καλύτερης απόδοσης.

Ο επόπτης πάνω στον οποίο έχει υλοποιηθεί ο μηχανισμός utmem είναι ο KVM, ο οποίος δεν υποστηρίζει αιτήματα tmem στη παρούσα του μορφή. Για αυτό το λόγο,

για τη δημιουργία του μηχανισμού υλοποιήθηκε αυτού του είδους η λειτουργικότητα. Αυτό επιτεύχθηκε προσθέτοντάς του μία κλήση επόπτη απέυθείας πάνω του. Το πλεονέκτημα της επιλογής αυτής είναι πως η υλοποίηση ήταν ιδιαίτερα απλή και εύκολη, και χρησιμοποιούσε ελάχιστο "boilerplate" και "glue" κώδικα. Οι προσθήκες στον κώδικα του επόπτη ήταν περίπου 150 γραμμές, οι οποίες προστέθηκαν σχεδόν δίχως τη μετατροπή του υπάρχοντος κώδικα. Η μόνη μετατροπή που αυτός χρειάστηκε ήταν η προσθήκη 3 γραμμών, ώστε να καλεί τις υπόλοιπες προσθήκες μας για τον κατάλληλο κωδικό κλήσης επόπτη.

Η κλήση επόπτη αυτή παραλαμβάνει μία περιοχή μνήμης από το guest, και δρα διαφορετικά ανάλογα με το αν το αίτημα ήταν GET ή PUT. Στην πρώτη περίπτωση, γεμίζει την περιοχή μνήμης με τα κατάλληλα δεδομένα που παραλαμβάνει από το πίσω μέρος μετά από κατάλληλο αίτημα, και επιστρέφει. Στη δεύτερη, αντιγράφει τα περιεχόμενα σε σε ένα νέο αντικείμενο, το οποίο τότε προσαρτάται στην περιοχή μνήμης tmem με ένα αίτημα PUT.

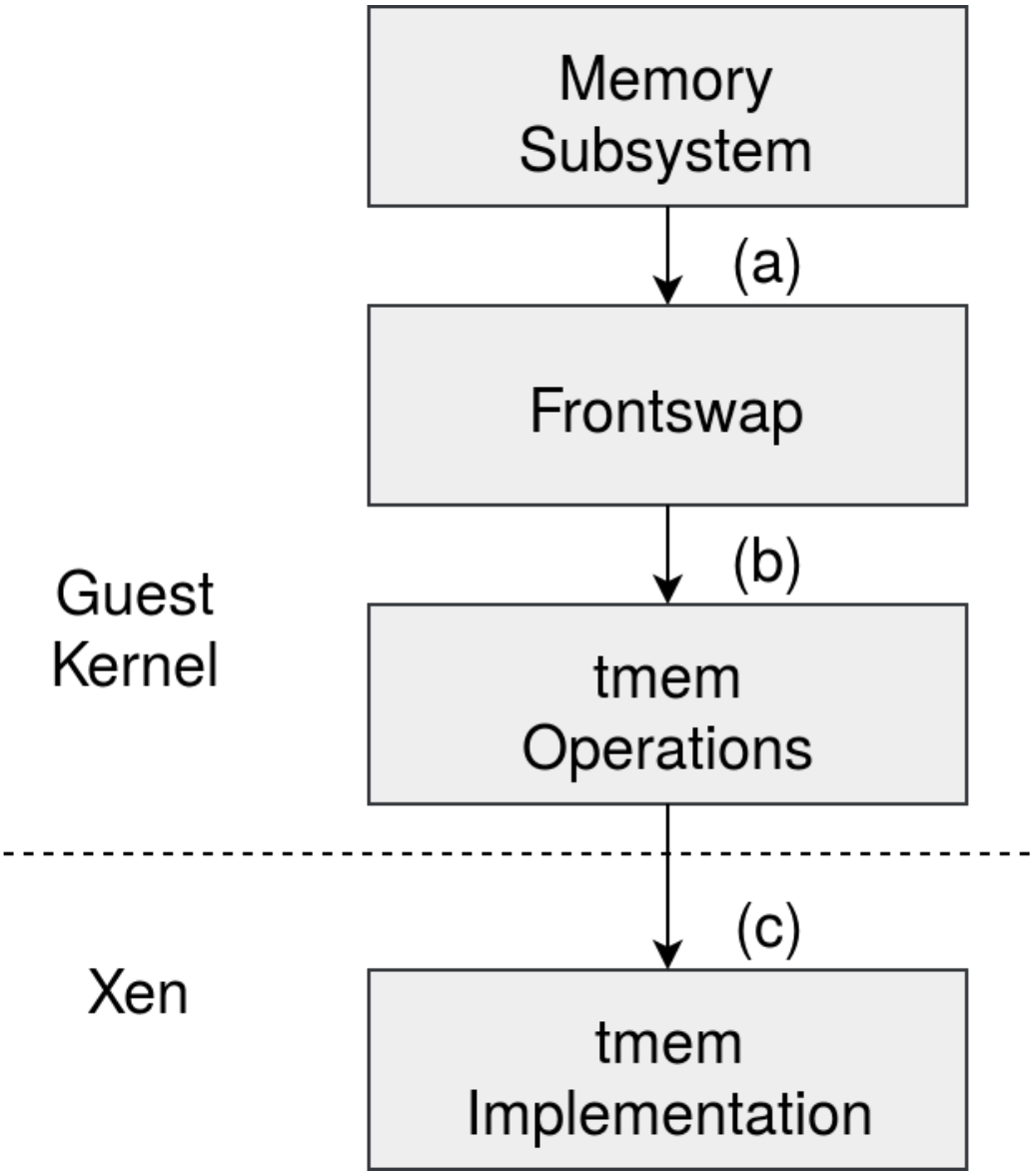
Η κεντρική ιδέα πίσω από αυτήν την αλληλουχία γεγονότων είναι η δημιουργία μίας αλυσίδας από χρήστες και πίσω μέρη utmem. Τα αιτήματα στην αρχή της αλυσίδας μεταφέρονται προς τα κάτω, μέχρι να βρουν ένα πίσω μέρος το οποίο μπορεί να αποδεχθεί ή να προσφέρει τα δεδομένα. Πιο συγκεκριμένα, το πίσω μέρος του guest απαρτίζεται τόσο από τον κώδικα ο οποίος εκτελεί μία κλήση επόπτη για κάθε αίτημα, όσο και από τον ίδιο τον επόπτη. Ο δεύτερος λειτουργεί με τη σειρά του ως χρήστης του τοπικού στο host πίσω μέρος, το οποίο μπορεί να εκτελέσει αναθέσεις μνήμης ώστε να αποθηκεύσει τα δεδομένα. Αφότου η τιμή του κλειδιού αποθηκευτεί, ενημερώνεται ο επόπτης, ο οποίος με τη σειρά του ενημερώνει τον αρχικό χρήστη μέσα στην εικονική μηχανή ο οποίος έκανε το αρχικό ερώτημα.

Το σύνολο μνήμης που χρησιμοποιείται δημιουργείται από ένα πίσω μέρος μηχανισμού tmem, το οποίο αποτελείται από έναν πίνακα κατακερματισμού ο οποίος γεμίζει με τα ζεύγη κλειδιού-τιμής τα οποία παρέχονται από την εικονική μηχανή, μαζί με επιπλέον κώδικα ο οποίος κρατά πληροφορίες για να επιβεβαιώση πως ο guest δεν καταναλώνει περισσότερη μνήμη του επιτρεπτού. Αν και έχει δημιουργηθεί ειδικά για το μηχανισμό utmem, το σύνολο μνήμης αυτό είναι απόλυτα συμβατό με υπάρχοντες μηχανισμούς tmem, και μπορεί να χρησιμοποιηθεί από το frontswap και το cleancache.

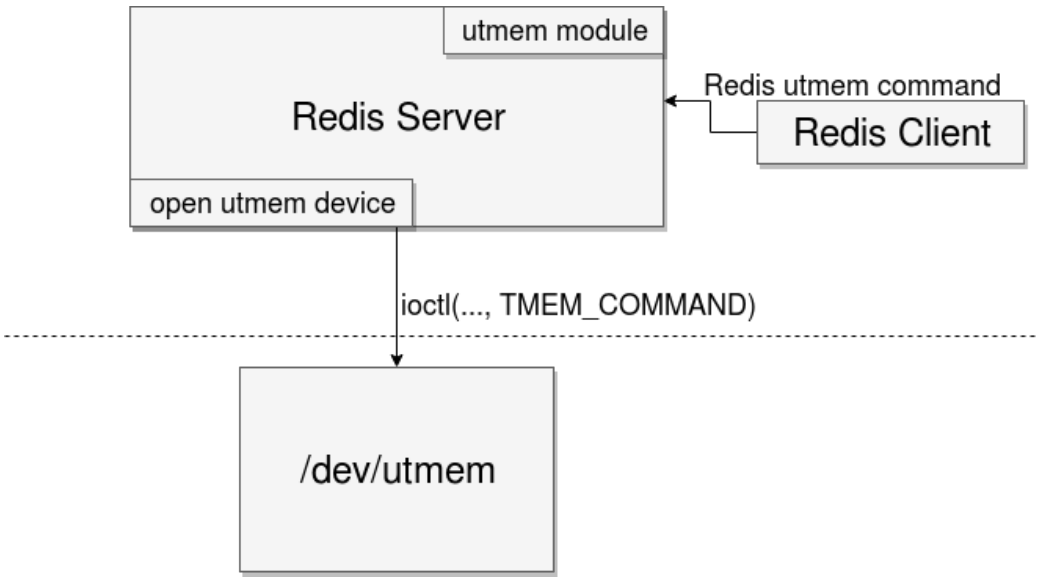
Επιλέξαμε ως εφαρμογή με την οποία θα μελετήσουμε την αποδοτικότητα του μηχανισμού το Redis, το οποίο είναι μία βάση κλειδιού-τιμής. Προφανώς, μία τέτοιου είδους εφαρμογή έχει API πολύ παρεμφερές αυτού του tmem. Η εφαρμογή επικοινωνεί με το μηχανισμό μέσα από μία συσκευή η οποία μπορεί να ελεγχθεί με χρήση της κλήσης συστήματος ioctl. Η συσκευή είναι υπεύθυνη για την αντιγραφή των ζευγών από το χώρο χρήστη στις περιοχές μνήμης την οποία χρησιμοποιεί ο πελάτης στο guest του μηχανισμού tmem, και ανάποδα.

Οι πιο σημαντικοί τύποι αιτημάτων οι οποίοι έχουν υλοποιηθεί είναι οι GET, PUT, INVALIDATE, οι οποίοι έχουν πανομοιότυπη χρήση με αυτή που έχουν στο συμβατικό μηχανισμό tmem. Οι πράξεις αυτές αντιστοιχούν σε αυτές οι οποίες υπάρχουν στη συντριπτική πλειονότητα των βάσεων κλειδιού-τιμής. Ως αποτέλεσμα, είναι εφικτή και λογική η σύγκριση της απόδοσης των ενεργειών του Redis, SET/GET, με αυτές του μηχανισμού μας, ώστε να συγκρίνουμε την απόδοσή του σε σχέση με τη βασική περίπτωση.

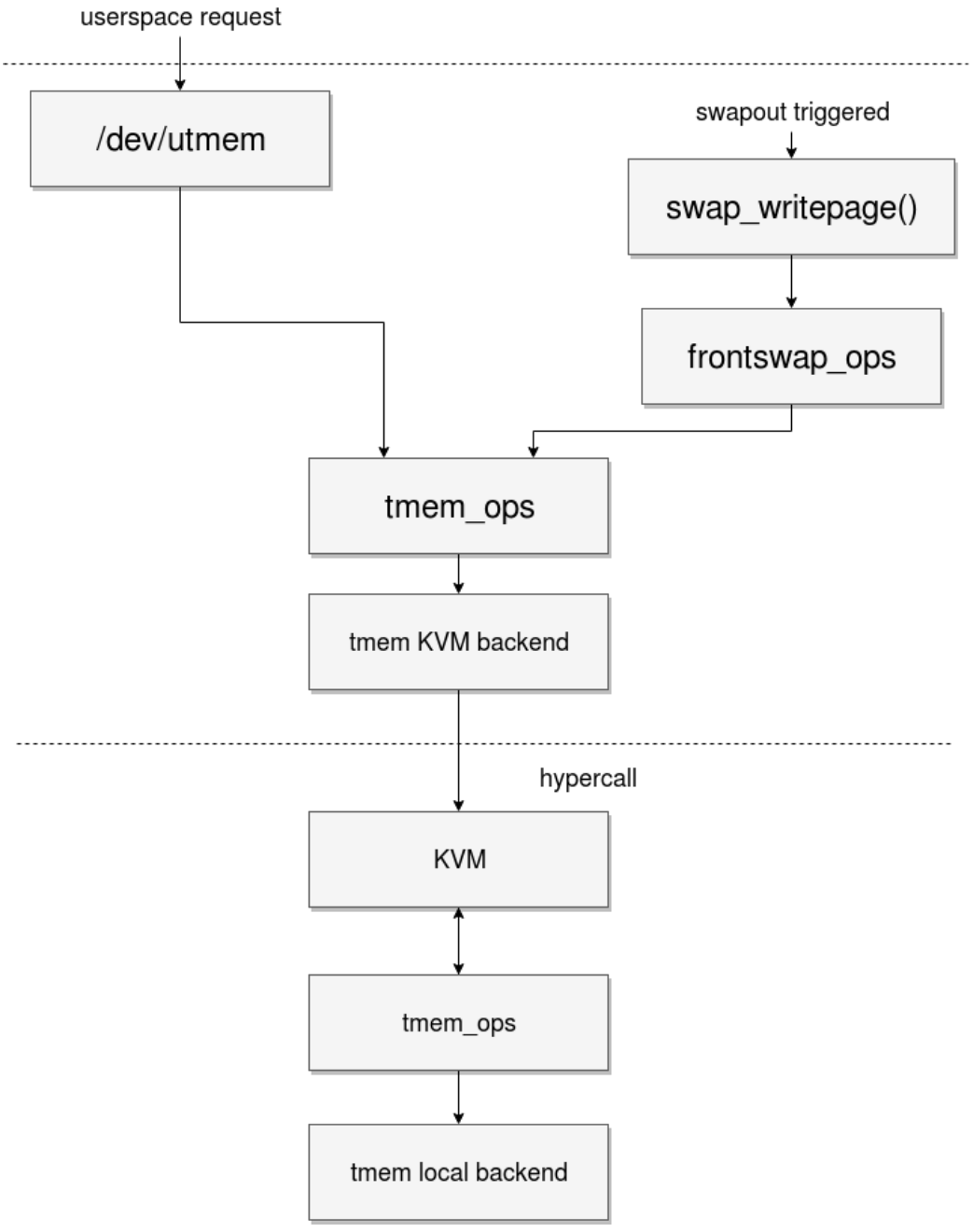
Η ακριβής αλληλουχία των κλήσεων οι οποίες γίνονται παρουσιάζεται στο σχήμα 2.4, το οποίο αναλύει τον ακριβή τρόπο με τον οποίο συνδέονται τα διάφορα στοιχεία του μηχανισμού. Το πρώτο μέρος του βρίσκεται εξ ολοκλήρου μέσα στην εικονική μηχανή, και έχει ως αποτέλεσμα την αποστολή μίας κλήσης επόπτη. Το δεύτερο μέρος βρίσκεται στο host, ξεκινά με τον επόπτη, και καταλήγει στο πίσω μέρος του μηχανισμού και το σύνολο μνήμης που χρησιμοποιείται. Τα σημεία εισόδου στο μηχανισμό μπορεί να βρίσκονται τόσο στο χώρο πυρήνα, όσο και στο χώρο χρήστη της εικονικής μηχανής.



Σχήμα 2.2: Οι στοιβάδες ενός συστήματος που μπορεί να χρησιμοποιήσει frontswap



Σχήμα 2.3: Το στρώμα συμβατότητας το οποίο επιτρέπει στο Redis να κάνει κλήσεις utmem



Σχήμα 2.4: Διάγραμμα κλήσεων του μηχανισμού utmem, τόσο για χρήστες στο χώρο πυρήνα όσο και στο χώρο χρήστη της εικονικής μηχανής

Κεφάλαιο 3

Μετρήσεις

3.1 Αξιολόγηση του συστήματος

Για την αξιολόγησή του συστήματός μας χρησιμοποιούμε το μετροπρόγραμμα (benchmark) του Redis, για τους προαναφερθέντες λόγους. Ο μηχανισμός utmem αξιολογείται σε σύο επίπεδα:

- Πόσο καλά λειτουργεί σε καταστάσεις όπου δεν υπάρχει καθόλου πίεση ως προς την ποσότητα διαθέσιμης μνήμης. Από τα αποτελέσματά μας διαπιστώνουμε μία ελαφρά πτώση της απόδοσης, λόγων των αντιγραφών δεδομένων τις οποίες εκτελεί ο μηχανισμός.

- Πόσο καλά λειτουργεί υπό πολύ μεγάλη (πάνω από 90% της μνήμης της εικονικής μηχανής εν χρήσει) πίεση μνήμης. Κατ΄ αυτόν τον τρόπο επιβεβαιώνουμε πως ο μηχανισμός utmem δεν παρουσιάζει τα γνωστά φαινόμενα τα οποία συνοδεύουν συνήθως τη χρήση τέτοιου ποσοστού της διαθέσιμης μνήμης, ενώ τόσο τα συμβατικά συστήματα όσο και αυτά τα οποία χρησιμοποιούν μόνο το μηχανισμό tmem έχουν σημαντικά μειωμένη απόδοση.

Αναλύουμε επίσης το χρόνο ο οποίος χρειάζεται για τα αιτήματα GET και PUT ανάλογα με το σε ποιο σημείο του μηχανισμού αναλώνεται. Χάρη στις μετρήσεις μας ανακαλύπτουμε, επίσης, και μια ασυμμετρία μεταξύ του κόστους των δύο αιτημάτων, η οποία πηγάζει από την υλοποίησή μας του μηχανισμού.

Το σύστημα στο οποίο εκτελέστηκαν τα πειράματα και πάρθηκαν οι μετρήσεις είναι ένας εξυπηρετητής με επεξεργαστή Intel Xeon X5650 και 48 GB RAM. Το λειτουργικό το οποίο χρησιμοποιήθηκε τόσο στον εκυπηρετητή όσο και στην εικονική μηχανή ήταν Ubuntu, μαζί με πυρήνα Linux 4.9. Για τη διαχείριση της εικονικής μηχανής χρησιμοποιήθηκε η βιβλιοθήκη libvirt, μαζί με το πρόγραμμα γραμμής εντολών virsh.

Πίνακας 3.1: Απόδοση των δύο αιτημάτων Redis (τιμές σε πράξεις/δευτερόλεπτο)

Value Size	SET	TMEMPUT
1KB	17121	15479
128KB	5569	4620
1024KB	2249	1964

3.1.1 Εκτίμηση της απόδοσης του utmem κατά την απουσία πίεσης μνήμης

Στον πρώτο γύρο πειραμάτων τα οποία εκτελέσαμε, επικεντρωθήκαμε στην απόδοση ενός συστήματος το οποίο δε χρειάζεται να καταφύγει στην εναλλαγή στο δίσκο. Πιο συγκεκριμένα, ελέγξαμε την απόδοση ενός μη τροποποιημένου εξυπηρετητή Redis με έναν ο οποίος μπορούσε να εκτελέσει κλήσεις tmem. Ο πελάτης Redis βρίσκεται στο host, έτσι ώστε η εκτέλεσή του να μην επηρεάζει την απόδοση της εικονικής μηχανής, αλλά και έτσι ώστε να βρίσκεται στο ίδιο φυσικό μηχάνημα ώστε η καθυστέρηση του δικτύου να μην αλλοιώνει τα αποτελέσματα. Κατά τα πειράματα δεν υπήρξε πίεση μνήμης, διότι χρησιμοποιούταν για κάθε αίτημα το ίδιο κλειδί. Το μετροπρόγραμμα το οποίο χρησιμοποιήθηκε ήταν αυτό του Redis.

Από το γράφημα του σχήματος 3.1, παρατηρούμε πως και οι δύο εξυπηρετητές παρουσιάζουν παρεμφερή ταχύτητα κατά την εξυπηρέτηση των αιτημάτων. Η διαφορά μεταξύ των δύο οφείλεται στις επιπλέον αντιγραφές τις οποίες εκτελεί ο μηχανισμός utmem. Η ταχύτητα του εξυπηρετητή ο οποίος χρησιμοποιεί utmem είναι περίπου το 85% αυτής του μη τροποποιημένου, ανεξαρτήτως του μεγέθους των δεδομένων τα οποία αποθηκεύονται. Αυτό είναι λογικό, καθώς η μόνη διαφορά ανάμεσα στις διαφορετικές περιπτώσεις είναι ο αριθμός κλήσεων επόπτη ανά μονάδα μεγέθους δεδομένων που μεταφέρεται από την εικονική μηχανή στο πίσω μέρος, και οι κλήσεις αυτές έχουν αμελητέο κόστος σε σχέση με τη μεταφορά, οπότε δεν προκαλούν αισθητή μεταβολή των μετρήσεων.

3.1.2 Ανάλυση της καθυστέρησης των αιτήσεων utmem

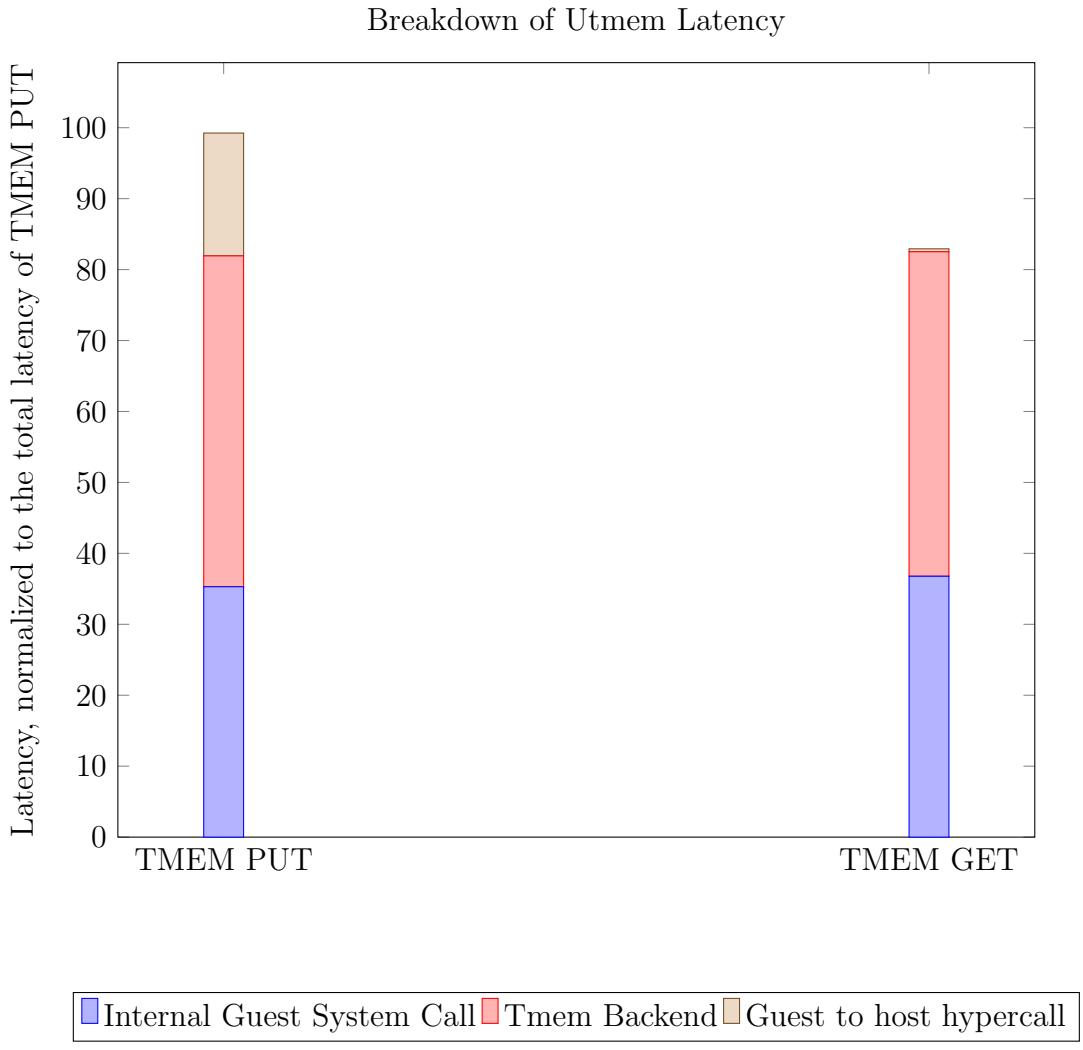
Για να μπορέσουμε να αποκτήσουμε μία καθαρότερη εικόνα σχετικά με τις πηγές καθυστέρησης των αιτημάτων utmem, δημιουργούμε ένα μικρομετροπρόγραμμα (microbenchmark) παρόμοιο αυτού του Redis. Η εργασία την οποία αυτό εκτελεί είναι να επαναλαμβάνει το ίδιο αίτημα για ένα δεδομένο αριθμό φορών, χρησιμοποιώντας μόνο ένα ζεύγος κλειδιού-τιμής μεγέθους 1024KB.

Σχήμα 3.1: Συγκριτική απόδοση του αρχικού με το τροποποιημένο με utmem σύστημα

Η σκιαγράφηση αυτή είναι εφικτή μέσα από την εκμετάλλευση των δυνατοτήτων που μας δίνει η συσκευή utmem να δημιουργήσουμε παραλλαγές των δύο αιτημάτων οι οποίες έχουν συγκεκριμένες ιδιότητα. Για την ακρίβεια, η συσκευή μπορεί να ρυθμιστεί ώστε να μην προωθεί αιτήματα στο πίσω μέρος του μηχανισμού, ή να απαντά σε κάθε αίτημα σαν αυτό να ήταν nop. Συνδυάζοντας και αντιπαραβάλλοντας την καθυστέρηση των εντολών για καθεμιά από αυτές τις παραλλαγές μπορούμε να διακρίνουμε πόσος χρόνος χρειάζεται σε οποιοδήποτε σημείο της διαδικασίας.

Στο σχήμα 3.2 φαίνονται τα αποτελέσματα του πειράματος. Όπως ακριβώς είχαμε υποθέσει, η συντριπτική πλειονότητα του κόστους (και για τις εγγραφές, το σύνολό

του) πηγάζει από τις αντιγραφές των δεδομένων, τόσο από την εφαρμογή στο χρήστη utmem, όσο και από τον τελευταίο στο πίσω μέρος του μηχανισμού. Ενδιαφέρον παρουσιάζει η ασυμμετρία μεταξύ των δύο αιτημάτων ως προς το χρόνο: Το PUT χρειάζεται να ξοδέψει κάποιο χρόνο στο πίσω μέρος, ενώ το GET όχι. Αυτό οφείλεται στο ότι το πρώτο αίτημα μπορεί να ικανοποιηθεί μόνο μετά από ανάθεση μνήμης για την αποθήκευση των δεδομένων, ενώ το δεύτερο έχει έτοιμες τις σχετικές περιοχές μνήμης και δε χρειάζεται να προβεί σε κάτι τέτοιο.



Σχήμα 3.2: Ανάλυση καθυστέρησης για τα αιτήματα utmem

Πίνακας 3.2: Αποτελέσματα της ανάλυσης καθυστέρησης των αιτημάτων (τιμές σε ms)

Operation	TMEMPUT	TMEMGET
Guest System Call	352.967	367.743
Tmem Backend	466.359	457.396
Guest to Host Hypercall	173.128	4.146

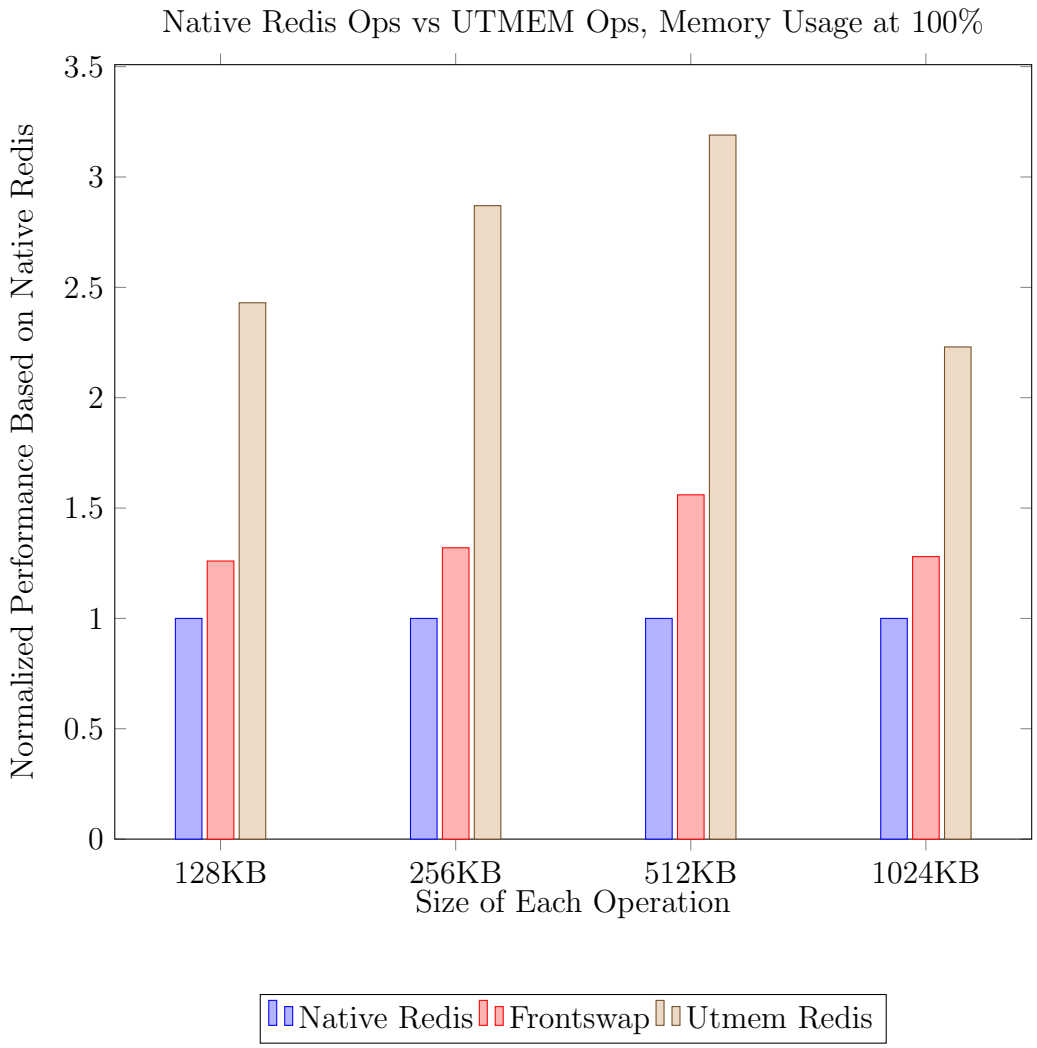
3.1.3 Αξιολόγηση του μηχανισμού utmem σε περιβάλλοντα με περιορισμένη μνήμη

Το επόμενο πείραμα το οποίο εκτελέσαμε αποτελεί και απόδειξη των πλεονεκτημάτων του μηχανισμού utmem απέναντι στις υπάρχουσες εναλλακτικές για συστήματα υπό πίεση μνήμης. Οι σχεδιασμοί απέναντι στους οποίους υπερτερεί συμπεριλαμβάνουν και αυτούς οι οποίοι χρησιμοποιούν tmem.

Κατά την εκτέλεση του πειράματος, γεμίζουμε τον εξυπηρετητή Redis ή το σύνολο μνήμης του πίσω τμήματος με τιμές οι οποίες έχουν συνολικό μέγεθος το 100% της μνήμης της εικονικής μηχανής. Ύστερα ξεκινούμε να εκτελούμε το μετροπρόγραμμα του Redis, σταματώντας μόνο όταν αυτό συγκλίνει.

Κοιτώντας τα αποτελέσματα στο Σχήμα 3.3 επιβεβαιώνουμε πως το σύστημα το οποίο κάνει χρήση του μηχανισμού utmem είναι ταχύτερο 2 με 3 φορές σε σχέση με το συμβατικό, ανάλογα με το μέγεθος των τιμών οι οποίες χρησιμοποιούνται. Το μη τροποποιημένο σύστημα υπόκειται σε τεράστια μείωση της απόδοσης λόγω της εναλλαγής σελίδων στο δίσκο, ακόμα και εάν δεν ξεπερνά το ποσό της μνήμης που του έχει ανατεθεί. Ένα ακόμα συμπέρασμα που βγάζουμε είναι πως το utmem υπερτερεί και του tmem, και πιο συγκεκριμένα των συστημάτων τα οποία το χρησιμοποιούν μέσω του frontswap για να μειώσουν το κόστος της εναλλαγής σελίδων. Έτσι, αποδεικνύεται πως η αποφυγή της επικοινωνίας με τη συσκευή μπλοκ δεν αποτελεί και εγγύηση πως θα διατηρηθεί η απόδοση.

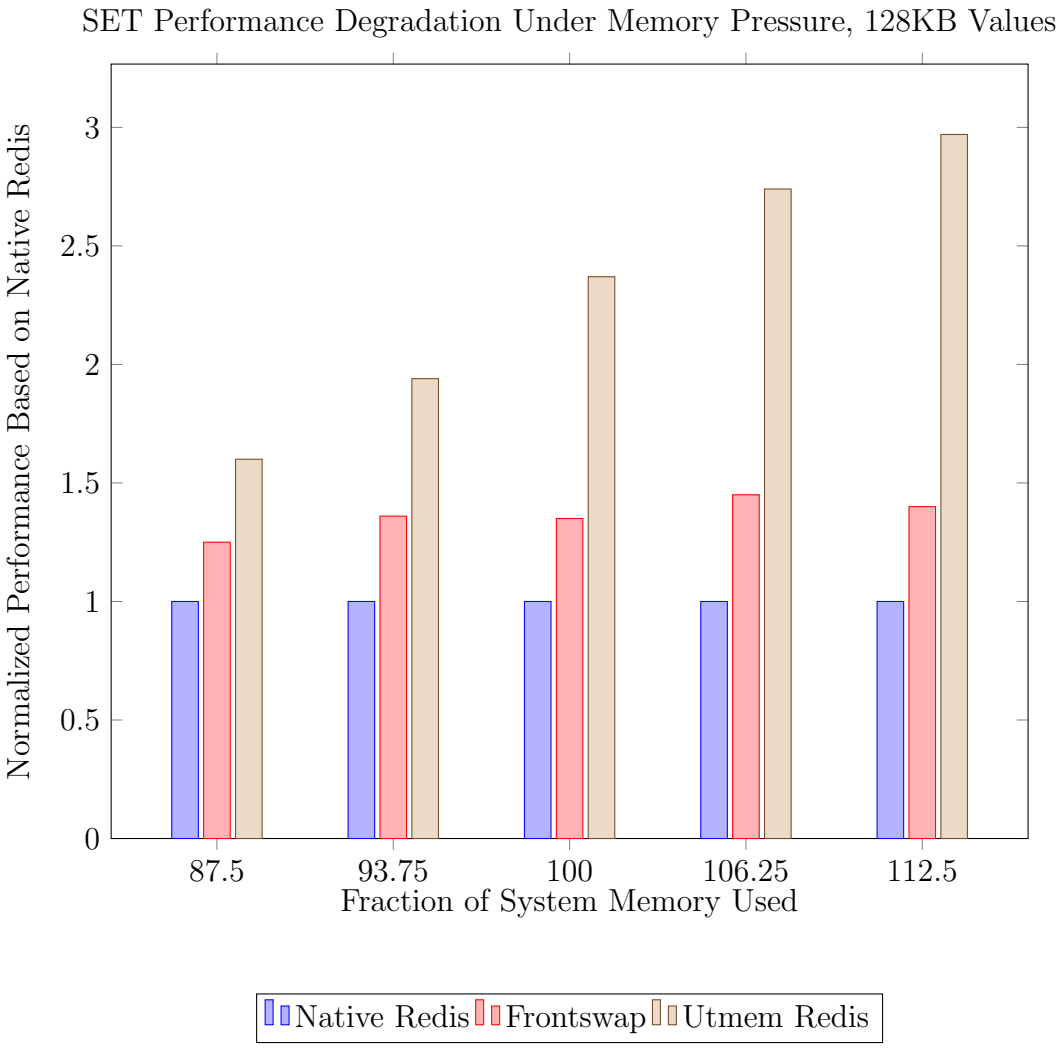
Μετρήσαμε επίσης την απόδοση της εικονικής μηχανής για κατανάλωση μνήμης διαφορετικού μεγέθους κάθε φορά, αλλά πάντα με το λόγο ως προς το ποσό διαθέσιμης μνήμης κοντά στη μονάδα. Βλέποντας τα αποτελέσματα στο Σχήμα 3.4 διαπιστώνουμε πως όσο μεγαλώνει η κατανάλωση μνήμης αυξάνεται και το πλεονέκτημα του μηχανισμού μας απέναντι στις εναλλακειτκές. Και πάλι, επίσης, φαίνεται πως το frontswap μειώνει και σταθεροποιεί, αλλά δεν εξαλειφει, το κόστος από πλευράς απόδοσης που έχει η μεγάλη κατανάλωση μνήμης.



Σχήμα 3.3: Απόδοση του μη τροποποιημένου συστήματος, του συστήματος που χρησιμοποιεί frontswap, και του συστήματος που χρησιμοποιεί tmem, για ποσοστό κατανάλωσης μνήμης ίσο με τη μονάδα

Πίνακας 3.3: Απόδοση εξυπηρέτησης αιτημάτων Redis για διαφορετικά μεγέθη τιμής, κατανάλωση μνήμης ύψους 100% (τιμές σε πράξεις ανά δευτερόλεπτο)

Value Size	TMEMPUT	Frontswap	SET
128KB	3739	1932	1536
256KB	2944	1354	1024
512KB	1829	896	572
1024KB	856	512	401



Σχήμα 3.4: Απόδοση του μη τροποποιημένου συστήματος, του συστήματος που χρησιμοποιεί frontswap, και του συστήματος που χρησιμοποιεί tmem, για μέγεθος τιμών 128KB

Πίνακας 3.4: Απόδοση εξυπηρέτησης αιτημάτων Redis για διαφορετικά μεγέθη καταναλισκώμενης μνήμης, τιμές μεγέθους 128KB (τιμές σε πράξεις ανά δευτερόλεπτο)

%memory pressure	TMEMPUT	Frontswap	SET
87.5	3823	2994	2389
93.75	3773	2638	1935
100	3697	2242	1655
106.25	3648	1933	1331
112.5	3626	1708	1219

Κεφάλαιο 4

Συμπεράσματα και Κατακλείδα

4.1 Σχετική Έρευνα

Το πιο σημαντικό παράδειγμα υλοποίησης μηχανισμού tmem είναι αυτό των Maggenheimer et al [19] για τον επόπτη Xen, με τον κώδικα να συνδέεται με προσθήκες οι οποίες έγιναν στον πυρήνα του Linux ακριβώς για αυτό το σκοπό. Περιλαμβάνει επίσης τους χρήστες της υλοποίησης, frontswap και cleancache, καθώς και ένα ειδικό πίσω μέρος ονόματι zswap το οποίο συμπιέζει τα δεδομένα πριν τα αποθηκεύσει. Η εργασία περιλαμβάνει επίσης μετρήσεις σχετικά με την υπεροχή του μηχανισμού σε σχέση με το ballooning.

Μία άλλη ενδιαφέρουσα εργασία η οποία σκιαγραφή μία πιθανή μελλοντική εφαρμογή του μηχανισμού tmem είναι αυτή του ex-tmem [23], η οποία χρησιμοποιεί μη πτητική μνήμη (nonvolatile memory) για το σύνολο μνήμης. Ο στόχος είναι η επαναχρησιμοποίηση του μοντέλου επικοινωνίας του μηχανισμού ώστε να προστεθεί ακόμα ένα στρώμα στην ιεραρχία μνήμης, με τη μη πτητική μνήμη ανάμεσα στη μνήμη τυχαίας προσπέλασης και το πίσω μέρος. Αυτό το μοντέλο προσφέρει με τη σειρά του τη δυνατότητα κατασκευής συστημάτων που επιδεικνύουν μεγαλύτερη ελαστικότητα μνήμης.

Άλλη μία εργασία η οποία αντλεί ιδέες από το μηχανισμό tmem είναι το σύστημα Mortar [24], το οποίο επικεντρώνεται σε σύνολα μνήμης παρεμφερή αυτών που απαρτίζονται από παροδική tmem, και το οποίο τα χρησιμοποιεί για να κατασκευάσει μοιραζόμενες κρυφές μνήμες οι οποίες εκμεταλλεύονται αχρησιμοποίητη μνήμη τόσο από το host όσο και από τις εικονικές μηχανές. Η μέθοδος αυτή παρουσιάζει πολλές ομοιότητες με το μηχανισμό utmem, καθώς και πάλι πρόκειται για μηχανισμούς tmem οι οποίοι χρησιμοποιούνται άμεσα από τις εφαρμογές.

Η διαφορά έγκειται στον τρόπο με τον οποίο η χρησιμοποιείται η επιπλέον μνήμη στα συστήματα: Στο δικό μας μηχανισμό αντί να ξεκινάμε με ένα μεγάλο σύνολο

μνήμης και να το μειώνουμε όταν χρειάζεται, έχομε ένα σύνολο το οποίο είναι όσο το δυνατόν μικρότερο και μεγαλώνει όταν αυτό είναι απαραίτητο. Αυτό οφείλεται στο ότι το Mortar χρησιμοποιεί παροδικά σύνολα μνήμης, και όχι μόνιμα. Επιπλέον, υπάρχει και ουσιαστική διαφορά στον τελικό στόχο των δύο μελετών, με το Mortar να αποσκοπεί στην εκμετάλλευση της αχρησιμοποίητης μνήμης, και το utmem να έχει ως στόχο τη συσσώρευση όσο περισσότερης μνήμης γίνεται στο host, ώστε η κατανομή της να ελέγχεται κεντρικά.

Μία άλλη σχεδιαστική επιλογή η οποία έχει ακολουθηθεί στις εργασίες αυτές είναι οι μηχανισμοί αυτοί να συνεργάζονται με μηχανισμούς ballooning, και να καταφεύγουν στη χρήση του μηχανισμού tmem για να απορροφήσουν απότομες αυξήσεις στην κατανάλωση μνήμης μέχρι ο πρώτος να προσαρμόσει τη μνήμη της εικονικής μηχανής κατάλληλα. Ο μηχανισμός tmem χρησιμοποιείται λοιπόν για τη διαχείριση παροδικών δεδομένων. Σε αντίθεση με την τακτική αυτή, ο μηχανισμός utmem αποσκοπεί στη χρήση των συνόλων μνήμης σε ημιμόνιμη βάση καθιστώντας τα αναπόσπαστο μέρος της ιεραρχίας μνήμης του συστήματος. Επιπλέον, η προσέγγισή μας επιτρέπει τη δυναμική αυξομείωση των ορίων της διαθέσιμης μνήμης, ενώ οι προαναφερθέντες μηχανισμοί δεν μπορούν από μόνοι τους να υπερβούν τις τιμές οι οποίες επιλέχθηκαν κατά τη δημιουργία της εικονικής μηχανής.

Πέραν αυτού του τύπου τους μηχανισμούς, έχουν υπάρξει και απόπειρες να γίνει ο μηχανισμός ballooning πιο γρήγορος στην απόκριση και τη λειτουργία του, έτσι ώστε να μη χρειάζεται να διατηρεί ένα ποσό μνήμης με το οποίο να απορροφά αυξήσεις στη ζήτηση για να αποφύγει τη μείωση της απόδοσης. Μία τέτοια προσέγγιση είναι η προσθήκη του μηχανισμού απέυθείας στο επίπεδο εφαρμογής [25], με εφαρμογές οι οποίες εκτελούν τη δική τους διαχείριση μνήμης όπως για παράδειγμα περιβάλλοντα εκτέλεσης γλωσσών προγραμματισμού να μπορούν άμεσα να χειριστούν το μηχανισμό. Μια εναλλακτική στρατηγική ακολουθείται στο μηχανισμό iBalloon [26], ο οποίος αντιμετωπίζει το αποδοτικό ballooning ως πρόβλημα μηχανικής μάθησης, και εφαρμόζει ανάλογες τεχνικές για να 'διδάξει' το μηχανισμό τη βέλτιστη ακολουθία βημάτων που πρέπει να ακολουθήσει για να φτάσει το σύστημα στα επιθυμητά επίπεδα μνήμης στον ελάχιστο χρόνο.

Η ιδέα της αφαίρεσης από το λειτουργικό σύστημα της εικονικής μηχανής της ευθύνης να διαχειριστεί τους πόρους της, έτσι ώστε αυτό να γίνεται κεντρικά για όλα τα μέρη του συστήματος, είναι ιδιαίτερα συχνή. Μία τέτοια εφαρμογή της είναι το Anykernel/Rump Kernel project [27], το οποίο μετατρέπει τον πυρήνα του λειτουργικού συστήματος NetBSD σε βιβλιοθήκη λειτουργικών συστημάτων (library OS) από το οποίο μπορούν να χρησιμοποιηθούν κομμάτια για τη σύνθεση ενός νέου πυρήνα, σχεδιασμένου να τρέχει μία μόνο εφαρμογή. Οι ειδικοί πυρήνες αυτοί, οι οποίοι αποκαλούνται rump kernels, αποτελούν ουσιαστικά συλλογές οδηγών συσκευών οι οποίες χρειάζονται τη βοήθεια του λειτουργικού συστήματος του host ακόμα και για τις βασικότερες ενέργειες, όπως είναι η δημιουργία διεργασιών ή η ανάθεση εικονικής

μνήμη. Πέρα από το χώρο του NetBSD, υπήρξε επίσης μία πρωτοβουλία με το όνομα Linux Kernel Library (LKL) [28], η οποία προσπαθεί να χρησιμοποιήσει τον πυρήνα του Linux για τον ίδιο σκοπό.

Ένα λιγότερο ακραίο παράδειγμα αφαίρεσης δυνατοτήτων από την εικονική μηχανή είναι αυτό των μονοπυρήνων. Αυτοί αποτελούν εξειδικευμένους πυρήνες με ελάχιστο κόστος εκτέλεσης, μεταγλωττισμένους με όλο το λογισμικό που χρειάζεται για την εκτέλεση μίας δεδομένης εφαρμογής, και μόνο με αυτό. Τα συστήματα μπορούν επίσης να εκτελέσουν μόνο βασικές ενέργειες ως προς τη διαχείριση της μνήμης τους, και βασίζονται στον επόπτη για τις υπόλοιπες [17]. Κάποιοι από τους πιο γνωστούς μονοπυρήνες είναι το Mirage OS, το οποίο είναι γραμμένο στη συναρτησιακή γλώσσα προγραμματισμού OCaml [17], και το OSv [29], το οποίο έχει ως σχεδιαστικό στόχο την εκτέλεση διεργασιών προορισμένων για συστήματα Linux σε περιβάλλοντα νέφους.

4.2 Μελλοντικές κατευθύνσεις

Αν και ο μηχανισμός ο οποίος παρουσιάστηκε στη εργασία αυτή είναι ήδη ανταγωνιστικός από πλευράς απόδοσης, υπάρχει μία πληθώρα βελτιστοποιήσεων οι οποίες μπορούν να εφαρμοστούν για την επίτευξη ακόμα καλύτερων αποτελεσμάτων. Δεδομένου ότι το κύριο πλεονέκτημα του utmem είναι η ικανότητά του να προσαρμόζεται εύκολα για χρήση από συγκεκριμένες εφαρμογές με ελάχιστες προσθήκες, οι βελτιστοποιήσεις οι οποίες προτείνονται παρακάτων έχουν περιστασιακή χρησιμότητα, και μπορούν ακόμα και να είναι αμοιβαία αποκλειόμενες.

Ένα τέτοιο παράδειγμα είναι η ύπαρξη διαφορετικών πίσω μερών στο host, το καθένα από τα οποία έχει διαφορετικά χαρακτηριστικά και είναι καταλληλότερο για διαφορετική χρήση. Για παράδειγμα, θα μπορούσε να υλοποιηθεί πίσω μέρος το οποίο να χρησιμοποιεί το δίκτυο για να αποθηκευσει της σελίδες σε απομακρυσμένο σύνολο μνήμης, όταν δεν υπάρχει διαθέσιμη μνήμη στο τοπικό μηχάνημα αλλά υπάρχει αλλού. Άλλη μία δυνατότητα είναι η προσθήκη ενός μηχανισμού deduplication στο σύστημα για τα αποθηκευμένα δεδομένα. Αφού οι τιμές μπορούν να είναι οντότητες διαφορετικές των συνηθισμένων (δηλαδή σελίδες), μπορούμε να εκμεταλλευτούμε την πολλαπλή εμφάνιση των ίδιων δεδομένων για να αυξήσουμε τα όρια χρησιμοποιούμενης μνήμης του συστήματος.

Παρόμοια ιδέα με τις παραπάνω είναι η εξής: Μπορεί να υπάρχουν εφαρμογές οι οποίες έχουν υψηλές απαιτήσεις από πλευράς κατανάλωσης μνήμης, όμως χαμηλές ανάγκες σε υπολογιστική ισχύ. Για να μειώσουμε την ανάγκη για τον πρώτο πόρο εκμεταλλευόμενοι το διαθέσιμο απόθεμά μας στο δεύτερο, θα ήταν εφικτό στους guests να συμπιέσουν ή γενικά να κωδικοποιήσουν τα δεδομένα έτσι ώστε να χρειάζονται λιγότερο χώρο στο σύνολο μνήμης. Η μέθοδος αυτή είναι παρεμφερής με το μηχανισμό

zswap [20], μόνο που εδώ η συμπίεση γίνεται από την πλευρά τση εφαρμογής, επιτρέποντας έτσι σε διαφορετικές εφαρμογές να χρησιμοποιήσουν διαφορετικούς αλγορίθμους για τα δεδομένα τους.

Καθώς ο μηχανισμός ο οποίος έχει περιγραφεί στην εργασία αυτή κοστίζει λίγο από πλευράς κύκλων επεξεργαστή, μπορεί αν χρησιμοποιηθεί στα πλαίσια εξειδικευμένων μηχανισμών οι οποίοι τροποποιούν σελίδες στη μνήμη χωρίς η εικονική μηχανή να έχει γνώση της ύπαρξης του σχετικού συνόλου μνήμης tmem. Στα συστήματα στα οποία μπορεί να εφαρμοστεί η ιδέα συμπεριλαμβάνονται τα περιβάλλοντα εκτέλεσης γλωσσών προγραμματισμού, τα οποία και είναι υπεύθυνα για μεγάλο μερος της διαχείρισης μνήμςη στους μονοπυρήνες για γλώσσες με συλλογή σκουπιδιών, καθώς και όλοι οι μηχανισμοί χώρου χρήστη οι οποίοι διαχειρίζονται μνήμη. Όσο υπάρχει τρόπος να παρεμβάλλουμε αιτήματα tmem ανάμεσα στις κλήσεις ενός API και στην πρόσβαση στα δεδομένα, όπως στο Redis, είναι εφικτό να χρησιμοποιηθεί εκεί ο μηχανισμός tmem. Λόγω της γενικότητας του μηχανισμού, μία πιο γενική υλοποίηση όπως για παράδειγμα ως παραεικονοποιημένη συσκευή virtio θα επέτρεπε την επαναχρησιμοποίηση κώδικα για όλα τα συστήματα τα οποία υποστηρίζουν τη δεδομένη κλάσση συσκευών.

Αν και η προσέγγισή μας είναι κατάλληλη για τη διαχείριση μεγάλου όγκου δεδομένων, κατά την αποθήκευση μικρότερου μεγέθους δομών το υπολογιστικό κόστος της κλήσης επόπτη μπορεί να γίνεται πιο αισθητό. Μια λύση στο πρόβλημα αυτό θα ήταν η σύμπτυξη αιτημάτων για μικρό μέγεθος δεδομένων σε μεγαλύτερα, αι η αποστολή τους στον επόπτη μόνο όταν μαζευτούν δεδομένα ενός δεδομένου μεγέθους. Η προσθήκη αυτή συνεπάγεται μιας 'κρυφής μνήμης' σε κάθε εικονική μηχανή, η οποία αποτρέπει την εκτέλεση εξόδων από αυτή κρατώντας τοπικά συχνά χρησιμοποιούμενες τιμές μικρού μεγέθους.

Άλλη μία πιθανή εφαρμογή του μηχανισμού θα ήταν στο live migration. Καθώς το σύνολο μνήμης στο πίσω μέρος δεν είναι άμεσα προσβάσιμο από την εικονική μηχανή, μπορεί να μετακινηθεί με διαφανή τρόπο χωρίς αυτή να το συνειδητοποιήσει. Αν κάποια δεδομένα τα οποία έχουν αποσταλλεί σε απομακρυσμένο σύνολο μνήμης ζητηθούν, μπορούν να ανασυρθούν από εκεί, με μόνο μειονέκτημα το επιπλέον κόστος του απομακρυσμένου αιτήματος. Το κομμάτι του μηχανισμού το οποίο επιλέγει το σύνολο μνήμης προς χρήση μπορεί να βρίσκεται στο host, όπως γίνεται στην υλοποίησή μας, οπότε η εικονική μηχανή δε χρειάζεται να γνωρίζει την ύπαρξη πολλαπλών συνόλων μνήμης. Αφού η μνήμη η οποία μπορεί να τροποποιηθεί ανά πάσα στιγμή περιορίζεται σε αυτήν η οποία είναι διευθυνσιοδοτήσιμη από την εικονική μηχανή, της οποίας το χώρο φυσικής μνήμης έχουμε περιορίσει, και το κύριο πρόβλημα της προσέγγισης pre-copy migration είναι οι αργοί χρόνοι σύγκλισης [30], ο μηχανισμός μας επιτρέπει την εφαρμογή της τελευταίας απαλλάσσοντάς την από το ζήτημα αυτό.

Όσον αφορά το συνδυασμό αποθηκευτικών χώρων μέσα στην εικονική μηχανή μαζί με το μηχανισμό utmem, άλλη μία βελτιστοποίηση θα ήταν η ανάπτυξη ευριστικών σύμφωνα με τις οποίες θα αποφασίζεται αν μία τιμή θα σταλεί στο μηχανισμό, ή θα παραμείνει στην εικονική μηχανή. Οι τεχνικές αυτές μπορούν να λαμβάνουν υπ'όψιν την τοπικότητα των δεδομένων, καθώς και την πίεση μνήμης τόσο στον guest όσο και στο host/

Μία ακόμα μετατροπή η οποία μπορεί υπό κατάλληλες συνθήκες να αποφέρει κέρδη ως προς την ταχύτητα είναι η χρήση κρυφών μνημών αντικειμένων (object caches) στο host ανάλογα με τη φύση των αντικειμένων που χρησιμοποιούνται. Κατάυτόν τον τρόπο, με αντάλλαγμα την ελαφρά αύξηση του μεγέθους της χρησιμοποιούμενης μνήμης μειώνεται το κόστος της προσθήκης εγγραφών, αφού δε χρειάζεται πια να γίνεται επιτόπου δέσμευση μνήμης.

Σε αυτήν την εργασία, έχουμε ασχοληθεί με την απόδοση του μηχανισμού utmem μόνο για μία εικονική μηχανή, και μόνο για τη μονοπύρηνη απόδοσή του. Καθώς η υλοποίηση μπορεί να γίνει πολυπύρηνη με τετριμμένες αλλαγές, μία λογική βελτιστοποίηση θα ήταν η μετατροπή της αυτή. Πέρα από την προφανή οδό προς την επίτευξη αυτής της λειτουργικότητας, θα μπορούσαν να γίνουν και πειραματισμοί με την προσθήκη πολυνηματισμού σε διάφορα επίπεδα του μηχανισμού, ώστε να διαπιστωθεί το σημείο στο οποίο αποδίδει βέλτιστα.

Με τους μονοπυρήνες να έχουν αποκτήσει ιδιαίτερο ερευνητικό ενδιαφέρον τα τελευταία χρόνια, μία πιθανή μελέτη αφορά την προσθήκη του μηχανισμού στα υποσυστήματα διαχείρισης μνήμης αυτών. Εάν αυτή η μετατροπή επιτύχει, το αποτέλεσμα θα είναι η έμμεση χρήση του από τις εφαρμογές, μέσω του κώδικα του μονοπυρήνα.

4.3 Κατακλείδα

Στην εργασία αυτή παρουσιάζουμε το μηχανισμό utmem, ο οποίος επεκτείνει τη δυνατότητα ενός συστήματος που χρησιμοποιεί εικονοποίηση να κάνει overcommit μνήμη στις εικονικές μηχανές του. Σχεδιάζουμε το μηχανισμό αυτό ακολουθώντας το παράδειγμα της υπάρχουσας αρχιτεκτονικής tmem, και επεκτείνοντάς την ώστε να είναι άμεσα χρησιμοποιήσιμη από το χώρο χρήστη. Αυτό μας επιτρέπει την επαναχρησιμοποίηση των υπάρχοντων χρηστών του tmem στο δικό μας πλαίσιο, καθώς και στη δημιουργία φορητών υλοποιήσεών του.

Για να μπορέσουμε να υποστηρίξουμε τη διαχείριση αιτημάτων tmem χωρίς τον επόπτη Xen, ο οποίος υλοποιεί το πίσω μέρος του μηχανισμού, δημιουργούμε μία εναλλακτική υλοποίηση η οποία μπορεί να τρέχει ανεξάρτητα στο λειτουργικό σύστημα Linux. Ύστερα, μετατρέπουμε τον επόπτη KVM έτσι ώστε να μπορούμε να μεταφέρουμε αιτήματα από την εικονική μηχανή στο πίσω μέρος το οποίο δημιουργήσαμε,

και παρέχουμε σε αυτήν ένα Linux module το οποίο μπορεί να κάνει κλήσεις επόπτη για να αποστέλλει αιτήματα. Ο μηχανισμός είναι προσβάσιμος από το χώρο χρήστη της μηχανής μέσα από τη χρήση μίας συσκευής η οποία μπορεί να δέχεται και να μεταδίδει αιτήματα.

Ο μηχανισμός μας αποδεικνύεται πως υπερτερεί σε συνθήκες αυξημένης κατανάλωσης μνήμης σε σχέση με μη τροποποιημένα συστήματα, αλλά και σε σχέση με συστήματα τα οποία χρησιμοποιούν υπαρκτούς μηχανισμούς tmem. Ακόμα και σε περιπτώσεις όπου υπάρχει άπλετη μνήμη στο σύστημα, όμως, η προσέγγισή μας παραμένει ανταγωνιστική.

Bibliography

[1] N. Amit, D. Tsafrir, and A. Schuster, "Vswapper: A memory swapper for virtualized environments," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 349–366, 2014.

[2] "QEMU Project Wiki." `https://wiki.qemu.org/Main_Page`. [Online; accessed 03-February-2018].

[3] "The Java Virtual Machine." `https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-1.html#jvms-1.2`. [Online; accessed 03-February-2018].

[4] "What is Docker?." `https://www.docker.com/what-container`. [Online; accessed 03-February-2018].

[5] "KVM Main Page." `https://www.linux-kvm.org/page/Main_Page`. [Online; accessed 03-February-2018].

[6] "The Xen Project." `https://www.xenproject.org/`. [Online; accessed 03-February-2018].

[7] N. Bhatia, "Performance evaluation of intel ept hardware assist," *VMware, Inc*, 2009.

[8] " Understanding Full Virtualization, Paravirtualization, and Hardware Assisted Virtualization." `https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf`. [Online; accessed 03-February-2018].

[9] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[10] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[11] " Intel Virtualization Technology ." `https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/`

intel-virtualization-technology.html. [Online; accessed 03-February-2018].

[12] " The Paravirtualization Spectrum, part 1: The Ends of the Spectrum ." https://blog.xenproject.org/2012/10/23/the-paravirtualization-spectrum-part-1-the-ends-of-the-spectrum/. [Online; accessed 28-April-2018].

[13] " The Paravirtualization Spectrum, part 2: From poles to a spectrum ." https://blog.xenproject.org/2012/10/31/the-paravirtualization-spectrum-part-2-from-poles-to-a-spectrum/. [Online; accessed 28-April-2018].

[14] " Xen Project Software Overview." https://wiki.xen.org/wiki/Xen_Project_Software_Overview#Guest_Types. [Online; accessed 1-May-2018].

[15] " GICv3 and GICv4 Software Overview ." http://infocenter.arm.com/help/topic/com.arm.doc.dai0492b/GICv3_Software_Overview_Official_Release_B.pdf. [Online; accessed 28-April-2018].

[16] J. Zhao and J. Pjesivac-Grbovic, "Mapreduce: The programming model and practice," 2009.

[17] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," in *Acm Sigplan Notices*, vol. 48, pp. 461–472, ACM, 2013.

[18] P. Sharma, *Page Cache Management in Virtual Environments*. PhD thesis, Indian Institute of Technology, Bombay Mumbai, 2012.

[19] D. Magenheimer, C. Mason, D. McCracken, and K. Hackel, "Transcendent memory and linux," in *Proceedings of the Linux Symposium*, pp. 191–200, Citeseer, 2009.

[20] S. Jennings, "Transparent memory compression in linux," 2013.

[21] D. Magenheimer *et al.*, "Transcendent memory on xen," *Xen Summit*, pp. 1–3, 2009.

[22] "Cleancache and Frontswap." https://lwn.net/Articles/386090/. [Online; accessed 26-February-2018].

[23] V. Venkatesan, W. Qingsong, and Y. Tay, "Ex-tmem: Extending transcendent memory with non-volatile memory for virtual machines," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on*, pp. 966–973, IEEE, 2014.

[24] J. Hwang, A. Uppal, T. Wood, and H. Huang, "Mortar: Filling the gaps in data center memory," *ACM SIGPLAN Notices*, vol. 49, no. 7, pp. 53–64, 2014.

[25] T.-I. Salomie, G. Alonso, T. Roscoe, and K. Elphinstone, "Application level ballooning for efficient server consolidation," in *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 337–350, ACM, 2013.

[26] J. Rao, X. Bu, K. Wang, and C.-Z. Xu, "iballoon: Self-adaptive virtual machines resource provisioning,"

[27] " Rump Kernels ." `http://rumpkernel.org/`. [Online; accessed 03-February-2018].

[28] O. Purdila, L. A. Grijincu, and N. Tapus, "Lkl: The linux kernel library," in *Roedunet International Conference (RoEduNet), 2010 9th*, pp. 328–333, IEEE, 2010.

[29] A. Kivity, D. L. G. Costa, and P. Enberg, "Os v—optimizing the operating system for virtual machines," in *Proceedings of USENIX ATC'14: 2014 USENIX Annual Technical Conference*, p. 61, 2014.

[30] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 273–286, USENIX Association, 2005.