Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Βελτιστοποίηση μεθόδων διαχείρισης φυσικής μνήμης για την υποστήριξη σύγχρονων ερευνητικών τεχνολογιών μετάφρασης διευθύνσεων

Διπλωματικη Εργασια

του

ΧΡΗΣΤΟΥ ΛΟΥΡΑ

Επιβλέπων: Γεώργιος Γκούμας
Αναπληρωτής Καθηγητής

Αθήνα, Σεπτέμβριος 2021

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Βελτιστοποίηση μεθόδων διαχείρισης φυσικής μνήμης για την υποστήριξη σύγχρονων ερευνητικών τεχνολογιών μετάφρασης διευθύνσεων

Διπλωματικη Εργασια

του

ΧΡΗΣΤΟΥ ΛΟΥΡΑ

Επιβλέπων: Γεώργιος Γκούμας
Αναπληρωτής Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28 Σεπτεμβρίου 2021.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος
Αναπληρωτής Καθηγητής	Καθηγητής	Καθηγητής

Αθήνα, Σεπτέμβριος 2021

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Χρήστος Λούρας

22 Σεπτεμβρίου 2021

Περίληψη

Η σημαντικότερη διαδικασία του συστήματος εικονικής μνήμης είναι η μετάφραση των εικονικών διευθύνσεων στις αντίστοιχες φυσικές και για την επιτάχυνσή της χρησιμοποιούνται διάφορες τεχνικές και μηχανισμοί, όπως οι πολυεπίπεδοι πίνακες σελίδες και τα TLBs. Ένα από τα σημαντικότερα άλυτα προβλήματα των σύγχρονων TLBs είναι η περιορισμένη κάλυψη των αντιστοιχίσεων του πίνακα σελίδων, το οποίο οδηγεί σε συχνή προσπέλαση του τελευταίου με αποτέλεσμα σημαντική χρονική επιβάρυνση. Για την βελτίωση της κάλυψης, υπάρχουν αρκετές σύγχρονες προτάσεις με κεντρική ιδέα την επέκταση του TLB ώστε με μία εγγραφή να μπορεί να μεταφράζει ένα αυθαίρετα μεγάλο εύρος συνεχόμενων εικονικών διευθύνσεων. Η διπλωματική αυτή εργασία διερευνά διαθέσιμους πειραματικούς μηχανισμούς υποστήριξης τέτοιων TLBs σε επίπεδο λειτουργικού συστήματος και προτείνει ένα νέο μηχανισμό που σκοπεύει σε μία καλύτερη επίτευξη μεταφραστικής συνέχειας συνδυάζοντας υπάρχουσες προτάσεις. Ο νέος μηχανισμός επεκτείνει τον μηχανισμό διαχείρισης μνήμης του Linux, ώστε να δημιουργεί συνεχόμενες αντιστοιχίσεις τόσο κατά την δέσμευση των σελίδων όσο και σε επόμενο χρόνο μετακινώντας ήδη δεσμευμένες σελίδες. Η αξιολόγηση του μηχανισμού, με την εκτέλεση διάφορων προγραμμάτων, έδειξε εξαιρετική κάλυψη της μνήμης με λίγες αντιστοιχίσεις ανεξάρτητα από την κατάσταση της φυσικής μνήμης και του βαθμού εξωτερικού κατακερματισμού της.

Λέξεις Κλειδιά

μεταφραστική συνέχεια, μετάφραση διευθύνσεων, TLB, εικονική μνήμη, διαχείριση μνήμης

Abstract

The most important operation of a virtual memory system is the translation of virtual addresses to the corresponding physical ones and various techniques and mechanisms are used for the acceleration of this operation such as as multi-layer page tables and TLBs. One of the worst unsolved problems of the modern TLBs is the limited coverage of the page table's mappings. This leads to frequent accessing of the page table and as a result it creates a serious overhead. In order to improve the TLB coverage, there are many modern proposals, whose idea is the expansion of TLB so as to translate an arbitrarily long contiguous virtual address range with a single mapping (range TLBs). This diploma thesis explores various available experimental software mechanisms, which can support range TLBs and it proposes a new mechanism, which aims to achieve better translation contiguity, based on combining existing proposals. The new mechanism expands Linux's memory management mechanism in order to create contiguous mappings during the allocation of pages and, afterwards, by moving misplaced pages to the correct memory spot. The evaluation of the mechanism, during the execution of various benchmarks, indicated exceptional memory coverage with very few mappings regardless of the state of physical memory and the level of its external fragmentation.

Keywords

translation contiguity, address translation, TLB, virtual memory, memory management

Ευχαριστίες

Καταρχάς θα ήθελα να ευχαριστήσω τον καθηγητή κ. Γεώργιο Γκούμα για την επίβλεψη αυτής της διπλωματικής εργασίας και για την ευκαιρία που μου έδωσε να την εκπονήσω στο εργαστήριο Υπολογιστικών Συστημάτων.

Επίσης, ευχαριστώ από καρδιάς την Χλόη Αλβέρτη, τον Στράτο Ψωμαδάκη και τον Βασίλη Καρακώστα, για την βοήθειά τους και την συνεργασία μας καθ όλη την διάρκεια της διπλωματικής εργασίας. Δίχως την καθοδήγησή τους, η εκπόνηση της εργασίας θα ήταν αρκετά δυσκολότερη και λιγότερο ευχάριστη.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και όλους τους φίλους και τις φίλες μου εκτός και εντός σχολής, που με βοήθησαν και με στήριξαν όλα αυτά τα χρόνια.

Αθήνα, Σεπτέμβριος 2021

Χρήστος Λούρας

Περιεχόμενα

Περίληψη **1**

Abstract **3**

Ευχαριστίες **5**

1 Εισαγωγή **13**
1.1 Δομή διπλωματικής εργασίας 15

2 Θεωρητικό Υπόβαθρο **17**
2.1 Εικονική Μνήμη 18
2.1.1 Πλεονεκτήματα-Μειονεκτήματα 18
2.1.2 Σελιδοποίηση 19
2.1.3 Πολιτικές δέσμευσης μνήμης 20
2.1.4 Μετάφραση διευθύνσεων 22
2.2 Μηχανισμός διαχείρισης μνήμης στο Linux 26
2.2.1 Μεταδεδομένα του μηχανισμού 26
2.2.2 Buddy Allocator 27
2.2.3 Transparent Huge Pages (THPs) 29
2.3 Πειραματικές επεκτάσεις μηχανισμών διαχείρισης μνήμης 30
2.3.1 Redundant Memory Mappings 30
2.3.2 Contiguity-aware Paging 33
2.3.3 Translation Ranger 37

3 Κίνητρο εργασίας **41**
3.1 Παράγοντες μειωμένης επίδοσης 41
3.1.1 Κατακερματισμός μνήμης 42
3.1.2 Μαζικές μετακινήσεις σελίδων 46
3.1.3 Μη-μετακινήσιμες σελίδες 46
3.2 Μία νέα πρόταση προς μία πιο ολοκληρωμένη επίτευξη συνέχειας 47
3.2.1 Διερεύνηση σεναρίου συνδυασμού των δύο μηχανισμών 47
3.2.2 Η πρότασή μας: ο μηχανισμός MACAP 48

4 Υλοποίηση **51**
4.1 Περιγραφή βασικού μηχανισμού 51
4.2 Μεταδεδομένα 51

4.3 Συντηρητική προσέγγιση στην δημιουργία νέων subVMAs 52
4.3.1 Περιγραφή Λειτουργίας 52
4.3.2 Παράγοντες αποτελεσματικότητας 52
4.3.3 Σκοπός 53
4.4 Μαρκάρισμα λανθασμένα τοποθετημένων σελίδων 53
4.5 Προληπτική δημιουργία νέου subVMA 53
4.5.1 Περιγραφή λειτουργίας 53
4.6 Μετακινήσεις σελίδων 55
4.7 Τροποποίηση του Buddy Allocator 55
4.8 Παραμετροποιησιμότητα μηχανισμού 55

5 Αξιολόγηση **57**
5.1 Μεθοδολογία 58
5.1.1 Περιβάλλον εκτέλεσης 58
5.1.2 Benchmarks 59
5.1.3 Μετρικές αξιολόγησης 60
5.1.4 Διαδικασία συλλογής αποτελεσμάτων 60
5.2 Πειράματα μετά από εκκίνηση μηχανήματος 61
5.2.1 Κάλυψη μνήμης 62
5.2.2 Δεσμεύσεις του CaP και μετακινήσεις σελίδων 62
5.2.3 Χρόνοι εκτέλεσης 63
5.3 Πειράματα με κατακερματισμένη μνήμη 64
5.3.1 Κάλυψη μνήμης 65
5.3.2 Δεσμεύσεις του CaP και μετακινήσεις σελίδων 70
5.3.3 Χρόνοι εκτέλεσης 72
5.4 Συμπεράσματα 77

6 Μελλοντικές επεκτάσεις **79**

7 Σχετική ερευνητική βιβλιογραφία **81**

Βιβλιογραφία **84**

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια **85**

Απόδοση ξενόγλωσσων όρων **87**

Κατάλογος Σχημάτων

2.1 Διαδικασία μετάφρασης διευθύνσεων σε αρχιτεκτονική x86-64 [1] 22
2.2 Πολυεπίπεδος πίνακας σελίδων (x86-64) [1] 24
2.3 Σχεδιασμοί για TLB 25
2.4 Σύγκριση κάλυψης ενός range TLB με τον κλασσικό TLB σελίδων 31
2.5 Ο πίνακας ευρών (range table) [2] 32
2.6 Αρχιτεκτονική διάταξη της μετάφρασης διευθύνσεων με χρήση και range TLB [2] 32
2.7 Εύρος συνεχόμενων αντιστοιχίσεων (contiguous mapping) [3] 33
2.8 Ο χάρτης συνέχειας (contiguity map) [3] 34
2.9 Βήματα επιλογής εύρους: 1)1ο page fault, 2) αναζήτηση στον χάρτη για ελεύθερη περιοχή, 3) επιλογή εύρους 4) ανανέωση δείκτη για επόμενη αναζήτηση. [3] 35
2.10 Παράδειγμα μετακίνησης σελίδων σε συνεχόμενες θέσεις [4] 37

3.1 Παράδειγμα αλλοίωση συνέχειας λόγω compaction. [4] 43
3.2 Επίδοση του **CaPaging** για διαφορετικά ποσοστά κατακερματισμού 44
3.3 Επίδοση του **Translation Ranger** για διαφορετικά ποσοστά κατακερματισμού 45

4.1 Διάγραμμα ροής τροποποιημένου CaPaging 54

5.1 Γραφήματα κάλυψης μνήμης σε καθαρή μη-κατακερματισμένη μνήμη . . . 61
5.2 Κάλυψη μνήμης για Micro. 66
5.3 Κάλυψη μνήμης με για Liblinear. 67
5.4 Κάλυψη μνήμης για XSBench. 68
5.5 Κάλυψη μνήμης για Hashjoin. 69
5.6 Δεσμέυσεις και μετακινήσεις μεγάλων σελίδων στο Micro. 70
5.7 Δεσμέυσεις και μετακινήσεις μεγάλων σελίδων στο Liblinear. 71
5.8 Δεσμέυσεις και μετακινήσεις μεγάλων σελίδων στο XSBench. 71
5.9 Δεσμέυσεις και μετακινήσεις μεγάλων σελίδων στο Hashjoin. 71
5.10 Χρόνοι εκτέλεσης για Micro. 73
5.11 Χρόνοι εκτέλεσης για XSBench. 74
5.12 Χρόνοι εκτέλεσης για Liblinear. 75
5.13 Χρόνοι εκτέλεσης για Hashjoin. 76

Κατάλογος Πινάκων

3.1 Μέσοι χρόνοι πειραμάτων 46
3.2 Πλήθος μετακινήσεων σελίδων λόγω Translation Ranger (σε GB) 46

5.1 Χαρακτηριστικά φυσικού μηχανήματος 58
5.2 Χαρακτηριστικά εικονικού μηχανήματος 58
5.3 Μέγεθος αποτυπώματος στην μνήμη και μέγιστο πλήθος VMAs ανά benchmark. 60
5.4 Πλήθος επιτυχημένων αρχικών τοποθετήσεων ανά μέγεθος σελίδας, σε καθαρή μνήμη. 62
5.5 Πλήθος μετακινήσεων σελίδων σε σενάριο καθαρής μνήμης 63
5.6 Μέσοι χρόνοι πειραμάτων σε κενή φυσική μνήμη 63
5.7 Πλήθος σελίδων ανά μέγεθος σε μη κατακερματισμένη μνήμη. 64

Κεφάλαιο 1

Εισαγωγή

Στην εποχή μας, τα υπολογιστικά συστήματα υψηλής επίδοσης (HPC) παρουσιάζουν μεγάλη και αυξανόμενη ζήτηση. Η χρήση υπερυπολογιστών είναι αναγκαία πια για περιπτώσεις όπως η εκπαίδευση νευρωνικών δικτύων, η μοντελοποίηση φυσικών φαινομένων και η επίλυση πολύπλοκων μαθηματικών προβλημάτων. Πέρα όμως από τις ερευνητικές χρήσεις, τα συστήματα υψηλής επίδοσης χρειάζονται και για βιομηχανικούς σκοπούς, όπως η δημιουργία μεγάλων βάσεων δεδομένων και η εξυπηρέτηση cloud υπηρεσιών.Για την ικανοποίηση όλων αυτών των χρήσεων, δεν απαιτούνται μόνο αρκετοί υπολογιστικοί πόροι (CPUs, GPUs, FPGAs) αλλά και αρκετή φυσική μνήμη. Όμως, για την επίτευξη καλών επιδόσεων δεν αρκεί μόνο ο πολλαπλασιασμός της διαθέσιμης φυσικής μνήμης, αλλά χρειάζεται και η ανανέωση και εξέλιξη των μηχανισμών διαχείρισης και αξιοποίησης της φυσικής μνήμης σε επίπεδο υλικού και λογισμικού (εικονική μνήμη).

Η εικονική μνήμη αποτελεί βασική τεχνική διαχείρισης της φυσικής μνήμης. Έχει καθιερωθεί ως αναπόσπαστο κομμάτι των σύγχρονων λειτουργικών συστημάτων και η καλή λειτουργία του μηχανισμού της είναι απαραίτητη για την καλή επίδοση ενός σύγχρονου μηχανήματος. Όμως, πέρα από τα οφέλη που μας προσφέρει ο μηχανισμός της εικονικής μνήμης, ένα από τα μειονεκτήματά της είναι η επιβάρυνση που δημιουργείται από την μετάφραση των εικονικών διευθύνσεων στις αντίστοιχες φυσικές διευθύνσεις.

Με σκοπό τον περιορισμό αυτής της χρονικής επιβάρυνσης, οι περισσότερες σύγχρονες υλοποιήσεις ενός μηχανισμού εικονικής μνήμης κάνουν χρήση της τεχνικής της σελιδοποίησης. Επίσης, σχεδιάστηκαν διάφοροι μηχανισμοί, όπως οι πολυεπίπεδοι πίνακες σελίδων και ο Translation Lookaside Buffer, οι οποίοι επιταχύνουν σημαντικά την αναζήτηση της φυσικής διεύθυνσης. Ειδικά, ο δεύτερος μηχανισμός (TLB) αποτελεί, σήμερα, αναπόσπαστο κομμάτι των επεξεργαστών και η χρησιμότητά του είναι αδιαμφισβήτητη.

Όμως, όσο αυξάνονται οι ανάγκες για περισσότερη φυσική μνήμη, δεν είναι εύκολο να μεγαλώνει αντίστοιχα και το TLB δίχως να γίνεται πιο αργό, διότι πρόκειται για μία ακόμα cache του επεξεργαστή. Μία πρώτη, ιστορικά, σημαντική λύση στο ζήτημα αυτό ήταν η δημιουργία των μεγάλων σελίδων (κυρίως 2MB στο x86-64) , όπου κάναν δυνατή την κάλυψη περισσότερου χώρου φυσικής μνήμης με τον ίδιο αριθμό καταχωρήσεων στο TLB.

Ακολούθησαν και άλλες μικρές ή μεγάλες βελτιώσεις (όπως τα Transparent Huge Pages στο λ.σ. Linux και οι 1GB μεγάλες σελίδες), αλλά δεν κατάφεραν να περιορίσουν αρκετά την χρονική επιβάρυνση στο TLB και γενικά στον μηχανισμό της εικονικής μνήμης, η οποία επιβάρυνση κυμαίνεται σήμερα στο 5-20% και σε εικονικοποιημένα περιβάλλοντα αγγίζει

και το 50% [1].

Ως αποτέλεσμα, τα τελευταία χρόνια έχει αναπτυχθεί έντονο ερευνητικό ενδιαφέρον για την αναζήτηση αποδοτικών λύσεων στην προσαρμογή των μηχανισμών διαχείρισης της φυσικής μνήμης στις τρέχουσες απαιτήσεις και, επίσης, για την σχεδίαση συστημάτων και μηχανισμών που θα μπορούν να ανταποκριθούν μακροπρόθεσμα στην όλο και αυξανόμενη ζήτηση για φυσική μνήμη.

Ένα σημαντικό τμήμα της πρόσφατης ερευνητικής έρευνας έχει ασχοληθεί με την σχεδίαση TLBs που χρησιμοποιούν μία καταχώρηση για την αποθήκευση πολλαπλών μεταφράσεων σελίδων (εν δυνάμει αρκετά μεγαλύτερων από μία μεγάλη σελίδα). Για την εκμετάλλευση των δυνατοτήτων των ερευνητικών TLBs απαιτείται η ύπαρξη μεγάλων συνεχών αντιστοιχίσεων σελίδας (contiguous page mappings), δηλαδή πολλές συνεχόμενες εικονικές σελίδες να αντιστοιχίζονται με συνεχόμενες φυσικές σελίδες. Όμως, τα σύγχρονα λειτουργικά συστήματα δεν διαθέτουν μηχανισμούς για την δημιουργία τέτοιων συνεχών αντιστοιχίσεων.

Πρόσφατα, έχουν προταθεί σχεδιασμοί που επεκτείνουν τις δυνατότητες του λειτουργικού συστήματος Linux προς αυτήν την κατεύθυνση. Δύο σχετικές προτάσεις είναι ο μηχανισμός Contiguity-aware Paging [3], που λειτουργεί κατά την εκχώρηση μνήμης από τον πυρήνα (allocation time), και ο μηχανισμός Translation Ranger [4], που τρέχει μέσω ενός νήματος πυρήνα το οποίο φτιάχνει μεγάλες συνεχείς αντιστοιχίσεις μετακινώντας τις υπάρχουσες σελίδες.

Η παρούσα διπλωματική εργασία έχει ως σκοπό την σύντομη παρουσίαση της κεντρικής ιδέας του πειραματικού TLB για μεγάλες συνεχείς αντιστοιχίσεις[2], την παρουσίαση και ανάλυση των δύο μηχανισμών που επεκτείνουν τις δυνατότητες του πυρήνα του Linux και, τέλος, τον συνδυασμό των δύο μηχανισμών με σκοπό την δημιουργία ενός αποδοτικότερου και πιο ολοκληρωμένου μηχανισμού δημιουργίας μεγάλων συνεχών αντιστοιχίσεων σελίδας.

1.1 Δομή διπλωματικής εργασίας

Η διπλωματική εργασία οργανώνεται ως εξής:

Στο **Κεφάλαιο 2** παρουσιάζεται διεξοδικά το θεωρητικό υπόβαθρο που χρειάζεται για την σωστή κατανόηση της διπλωματικής εργασίας. Αρχικά, περιγράφεται η ιδέα της εικονικής μνήμης, το πως επηρεάζει θετικά και αρνητικά τους διάφορους μηχανισμούς διαχείρισης μνήμης που την χρησιμοποιούν και κάποια θεμελιώδη τμήματα των υποσυστημάτων εικονικής μνήμης. Ακολουθεί μία σύντομη ανάλυση διαφόρων τμημάτων του μηχανισμού διαχείρισης μνήμης του λειτουργικού συστήματος Linux, χρήσιμα για την κατανόηση των σχεδιαστικών επιλογών που έγιναν κατά την κατασκευή του νέου μηχανισμού. Τέλος, παρουσιάζονται εκτενώς διάφορες ερευνητικές προτάσεις για την βελτίωση των μηχανισμών διαχείρισης μνήμης σε ζητήματα τα οποία αποτελούν και το ερευνητικό ενδιαφέρον αυτής της διπλωματικής εργασίας.

Στο **Κεφάλαιο 3** παρουσιάζεται η συλλογιστική μας πορεία που μας οδήγησε στην υλοποίηση του συνδυασμού των μηχανισμών Contiguity-aware Paging και Translation Ranger. Αρχικά, παρουσιάζονται διάφοροι παράγοντες που οδηγούν σε κακή επίδοση των υπάρχοντων μηχανισμών με συνοδεία κάποιων παρατηρήσεών μας από πειράματα. Με την εκτενή ανάλυση των παραγόντων γίνεται μία προσπάθεια να εξηγηθεί το γιατί ο συνδυασμός των δύο αυτών μηχανισμών αποτελεί μία δόκιμη και αισιόδοξη κατεύθυνση προς ένα καλύτερο και πιο ολοκληρωμένο μηχανισμό επίτευξης μεταφραστικής συνέχειας. Τέλος, αναφερόμαστε στις απαιτήσεις που έχουμε από τον νέο μηχανισμό και στην συνέχεια γίνεται μία μικρή εισαγωγή στον νέο μηχανισμό, που τον ονομάσαμε MACAP (Migration-assisted Contiguity-aware Paging).

Στο **Κεφάλαιο 4** περιγράφεται αναλυτικά η υλοποίηση του νέο μηχανισμού. Γίνεται εκτενής αναφορά στα νέα μεταδεδομένα του μηχανισμού και στις χρήσιμες νέες δυνατότητες στον μηχανισμό του Contiguity-aware Paging. Τέλος, παρουσιάζεται ο ρόλος του τροποποιημένου Translation Ranger στον νέο μηχανισμό και κάποιες λίγες πληροφορίες σχετικά με τις δυνατότητες παραμετροποίησης της λειτουργίας του νέου συνδυασμού και κάποιες νέες αναγκαίες τροποποιήσεις στον buddy allocator.

Στο **Κεφάλαιο 5**, αφού αναφερθούμε στο περιβάλλον εκτέλεσης των πειραμάτων μας για τον νέο μηχανισμό, παρουσιάζουμε τα αποτελέσματά των πειραμάτων συγκριτικά με τους υπόλοιπους μηχανισμούς. Τέλος, αξιολογούμε την προσέγγισή μας σε αυτήν την διπλωματική εργασία και συγκρίνουμε συνολικά την συμπεριφορά των μηχανισμών.

Στο **Κεφάλαιο 6** παρουσιάζουμε μερικές ιδέες για μελλοντική ενασχόληση με τον μηχανισμό που παρουσιάσαμε.

Στο **Κεφάλαιο 7** αναφέρουμε πρότερη ερευνητική βιβλιογραφία πάνω στο ζήτημα της μειούμενης κάλυψης των σύγχρονων TLBs και στον επίτευξη μεταφραστικής συνέχειας.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

Όπως αναφέρθηκε στην Εισαγωγή, στο κεφάλαιο αυτό θα παρουσιαστούν τα θεμέλια πάνω στα οποία βασίστηκε η εργασία της διπλωματικής εργασίας. Αρχικά, γίνεται εκτενής αναφορά στην ιδέα της εικονικής μνήμης και σε θεμελιώδη υποτμήματα των υποσυστημάτων εικονικής μνήμης. Ασχολούμαστε με την όσο καλύτερη επεξήγηση ζητημάτων, όπως η σελιδοποίηση και η μετάφραση διευθύνσεων, που αποτελούν βασικά θεμέλια όλης της ερευνητικής σχετικής βιβλιογραφίας και του θέματος της διπλωματικής εργασίας.

Στην συνέχεια, αναλύονται τμήματα του μηχανισμού διαχείρισης μνήμης του λ.σ. Linux, τα οποία σχετίζονται με τις ερευνητικές προτάσεις που αναφέρονται και του μηχανισμού που αποτελεί το προϊόν της παρούσας εργασίας.

Τέλος, παρουσιάζεται πρότερη σχετική έρευνα πάνω στην εικονική μνήμη και την κάλυψη του TLB. Συγκεκριμένα, παρουσιάζονται αρχικά η προσέγγιση των Redundant Memory Mappings [2] και στην συνέχεια οι μηχανισμοί υποστήριξης σε επίπεδου λογισμικού, Contiguity-aware Paging[3] και Translation Ranger[4].

2.1 Εικονική Μνήμη

Η εικονική μνήμη αποτελεί μια θεμελιώδης αφαίρεση ενός υπολογιστικού συστήματος και αφορά τους διαθέσιμους πόρους αποθήκευσης ενός συστήματος [1]. Ειδικά στα σύγχρονα λειτουργικά συστήματα, η εικονική μνήμη αποτελεί θεμελιώδες χαρακτηριστικό τους και τα προγράμματα που εκτελούνται σε αυτά πραγματοποιούν προσβάσεις στην μνήμη με την χρήση εικονικών διευθύνσεων. Μπορούμε να φανταστούμε την εικονική μνήμη ως ένα API για την χρήση της φυσικής μνήμης των μηχανημάτων και γενικά των αποθηκευτικών χώρων από τα προγράμματα.

Τα προγράμματα, όπως προαναφέρθηκε, για την προσπέλαση της φυσικής μνήμης έχουν μόνο την δυνατότητα χρήσης ενός συνεχούς χώρου εικονικών διευθύνσεων. Κάθε πρόγραμμα έχει το δικό του τέτοιο χώρο μνήμης και δεν χρειάζεται να αντιστοιχεί στα χαρακτηριστικά του φυσικού χώρου μνήμης που χρησιμοποιεί και στο οποίο τρέχει. Δηλαδή, ο εικονικός χώρος μνήμης μπορεί να έχει και το μέγιστο θεωρητικό εύρος μνήμης που μπορεί να αναγνωρίσει το λειτουργικό σύστημα στην εκάστοτε αρχιτεκτονική και δεν επηρεάζεται από την αλληλεπίδραση άλλων προγραμμάτων με την μνήμη και τη τρέχουσα χρήση της φυσικής μνήμης.

Η μετατροπή των εικονικών διευθύνσεων στις αντίστοιχες φυσικές διευθύνσεις αποτελεί ευθύνη του λειτουργικού συστήματος με την βοήθεια εξειδικευμένων μηχανισμών των επεξεργαστών (TLB cache). Επίσης, το λειτουργικό σύστημα είναι υπεύθυνο και για την εκχώρηση μνήμης σε ένα πρόγραμμα και την κατάλληλη αναζήτηση θέσεων φυσικής μνήμης και συσχέτισή τους με τις αντίστοιχες εικονικές διευθύνσεις του προγράμματος. Τέλος, η ταχύτητα και η αδιαβλητότητα αυτού του μηχανισμού είναι πάρα πολύ σημαντική για την εύρυθμη λειτουργία ενός μηχανήματος. Ο μηχανισμός πρέπει να εκτελεί γρήγορα τις ενέργειές του, διότι ενεργοποιείται συνεχώς, αλλά συγχρόνως πρέπει να κάνει και όλους τους αναγκαίους ελέγχους για την αποφυγή προβλημάτων ασφάλειας και διαθεσιμότητας φυσικού χώρου.

2.1.1 Πλεονεκτήματα-Μειονεκτήματα

Στην συνέχεια, παρουσιάζονται διάφορα πλεονεκτήματα από την χρήση της εικονικής μνήμης:

1. Παρέχει προστασία δεδομένων και απομόνωση σε μία εφαρμογή, εμποδίζοντας ελαττωματικό ή κακόβουλο κώδικα να προσπελάσει περιοχές της φυσικής μνήμης που ανήκουν στο λειτουργικό σύστημα ή σε άλλες εφαρμογές.

2. Απελευθερώνει τις εφαρμογές από το βάρος της διαχείρισης ενός κοινού φυσικού χώρου μνήμης. Με την χρήση της εικονικής μνήμης, μία εφαρμογή δεν χρειάζεται να την απασχολούν οι υπόλοιπες ενεργές εφαρμογές και το πως είναι αποθηκευμένα τα δεδομένα τους στην φυσική μνήμη. Έτσι, επιπλέον γίνεται λιγότερο πολύπλοκη και πιο γρήγορη η ανάπτυξη προγραμμάτων.

3. Ένα ακόμα πλεονέκτημα, που αν και σχετίζεται με το παραπάνω αξίζει να σχολιαστεί και αναλυθεί ξεχωριστά, είναι το γεγονός ότι η ύπαρξη της εικονικής μνήμης βοηθάει στην καλύτερη ανάπτυξη φορητού κώδικα. Επιτρέπει στους προγραμματιστές να

μην λαμβάνουν υπόψιν τις ιδιαιτερότητες (πχ όρια, μέγεθος) της φυσικής μνήμης του συστήματος στο οποίο θα τρέξει και καθιστά την καλή κατανόηση των λειτουργιών χαμηλού επιπέδου του συστήματος προαιρετική για την ανάπτυξη ενός προγράμματος που θα τρέχει αξιοπρεπώς σε διαφορετικά μηχανήματα.

4. Η εικονική μνήμη κάνει ευκολότερη και αποδοτικότερη την κατανομή και διαχείριση της φυσικής μνήμης. Οι εφαρμογές ασχολούνται μόνο με τον εικονικό χώρο διευθύνσεών τους και δεν επηρεάζονται λειτουργικά από αλλαγές που γίνονται στο επίπεδο της φυσικής μνήμης. Αυτό επιτρέπει την ευκολότερη και αποτελεσματικότερη εκτέλεση διαφόρων διεργασιών στην μνήμη, όπως την μετακίνηση σελίδων σε άλλα αποθηκευτικά μέσα ή άλλες θέσεις στην φυσική μνήμη. Επίσης, η αφαίρεση της εικονικής μνήμης επιτρέπει την χρήση πολύπλοκων μηχανισμών, όπως το demand paging, το swapping και το defragging της μνήμης (θα αναλυθούν εκτενώς στην συνέχεια).

5. Κάνει εύκολη την κοινή χρήση δεδομένων ή βιβλιοθηκών μεταξύ εφαρμογών. Τα κοινά δεδομένα βρίσκονται σε ένα αντίγραφο στην φυσική μνήμη, αλλά κάθε εφαρμογή μπορεί να τα προσπελάσει αυτόνομα μέσω του εικονικού χώρου διευθύνσεών της. Με άλλα λόγια, οι εφαρμογές δεν χρειάζονται να υλοποιήσουν κάποια ειδική μεταχείρισή τους και αυτός ο ρόλος συνήθως ανήκει στο λειτουργικό σύστημα.

Τα προτερήματα από την εκμετάλλευση μηχανισμών εικονικής μνήμης είναι, αναμφίβολα, σημαντικότατα. Όμως, η χρήση εικονικής μνήμης δεν έρχεται δίχως κάποια μειονεκτήματα:

1. Η μετάφραση των διευθύνσεων επιβαρύνει χρονικά την εκτέλεση του προγράμματος. Κατά την εκτέλεση του προγράμματος, καταναλώνονται πόροι για την μετάφραση των εικονικών διευθύνσεων.

2. Οι συσχετίσεις των εικονικών και φυσικών διευθύνσεων κάπου πρέπει να αποθηκεύονται και μπορεί να καταλαμβάνουν πολύτιμο χώρο στα αποθηκευτικά μέσα. Επίσης, επιβαρύνουν την διαδικασία της εναλλαγής διεργασιών στον επεξεργαστή (context switching).

2.1.2 Σελιδοποίηση

Τα περισσότερα σύγχρονα υπολογιστικά συστήματα στο υποσύστημα εικονικής μνήμης, σε επίπεδο λογισμικού και υλικού, κάνουν χρήση της τεχνικής της σελιδοποίησης (paging)[1]. Ο χώρος εικονικών διευθύνσεων χωρίζεται σε τμήματα συγκεκριμένου μεγέθους, που ονομάζονται σελίδες (pages) και ο χώρος φυσικών διευθύνσεων σε αντίστοιχα τμήματα που ονομάζονται πλαίσια (page frames). Το υποσύστημα εικονικής μνήμης διαχειρίζεται κυρίως την μνήμη σε επίπεδο σελίδων. Δηλαδή, όλες οι μετακινήσεις δεδομένων (π.χ. RAM προς τον δίσκο αποθήκευσης) γίνονται μετακινώντας ολόκληρες σελίδες, ενώ και η μετάφραση διευθύνσεων γίνεται σε επίπεδο σελίδων και όχι μεμονωμένων διευθύνσεων. Με άλλα λόγια, η μικρότερη μονάδα στο υποσύστημα εικονικής μνήμης είναι η σελίδα/πλαίσιο.

Πλεονεκτήματα-Μειονεκτήματα

Αποφεύγοντας μία πιο fine-grained προσέγγιση, με την βοήθεια της σελιδοποίησης, μειώνεται αρκετά η επιβάρυνση κατά την μετάφραση διευθύνσεων και ο χώρος που δεσμεύεται για τις αντιστοιχίσεις. Όσο μεγαλύτερο είναι το μέγεθος της σελίδας βάσης (4KB στο x86-64), τόσο μεγαλύτερη κάλυψη προσφέρουν μηχανισμοί, όπως ο TLB, οι οποίοι δεν μπορούν να αυξήσουν σημαντικά το μέγεθός του δίχως να θυσιαστεί μέρος της απόδοσής τους. Επίσης, η σελιδοποίηση οδηγεί σε μικρότερο δεσμευμένο χώρο για μεταδεδομένα, διότι πολλές θέσεις μνήμης αντιμετωπίζονται σαν ένα ομογενές block με ίδια χαρακτηριστικά, ίδιο ¨ιδιοκτήτη¨ και ίδια δικαιώματα ανάγνωσης/εγγραφής.

Ένα αρνητικό της σελιδοποίησης είναι ότι μπορεί να δημιουργήσει εσωτερικό κατακερματισμό (internal fragmentation). Αυτό συμβαίνει διότι τμήματα σελίδων που έχουν δεσμευτεί μπορεί να μένουν αναξιοποίητα από τον ¨κάτοχο¨ των σελίδων και δεν γίνεται τα ελεύθερα τμήματα των σελίδων να δοθούν σε άλλη διεργασία. Για να περιοριστεί το φαινόμενο του εσωτερικού κατακερματισμού, επιλέγεται ένα σχετικά μικρό μέγεθος για την σελίδα βάσης και οι μηχανισμοί διαχείρισης μνήμης των λ.σ. προσφέρουν δυνατότητες βέλτιστης κάλυψης των δεσμευμένων, για τις ανάγκες του πυρήνα, σελίδων (π.χ. slab/slub/slob allocators στον πυρήνα Linux).

Μεγάλες Σελίδες

Η σύλληψη της ιδέας των μεγάλων σελίδων προήλθε από την αναζήτηση λύσεων για την αύξηση της κάλυψης του TLB και την μείωση των αποτυχημένων αναζητήσεων σε αυτό, οι οποίες όλο και αυξάνονται με την παράλληλη αύξηση της διαθέσιμης φυσικής μνήμης. Οι μεγάλες σελίδες αποτελούν συνεχόμενα block μνήμης, μεγαλύτερου (πολλαπλάσιου) μεγέθους από αυτό την σελίδας βάσης, που αντιμετωπίζονται με τον ίδιο τρόπο που αντιμετωπίζεται μία σελίδα βάσης, αλλά επιτρέπουν την μεγαλύτερη κάλυψη από το TLB με τον ίδιο αριθμό καταχωρήσεων.

Η χρήση μεγάλων σελίδων δεν αποτελεί καθολική και σίγουρη λύση στο παραπάνω πρόβλημα. Σε ερευνητικά προγράμματα και προγράμματα μαθηματικών υπολογισμών, προβλέπεται βελτίωση της επίδοσης έως και 45% [5], ενώ σε βάσεις δεδομένων και παρόμοια προγράμματα συνεχούς δέσμευσης και αποδέσμευσης μνήμης δεν βοηθά αρκετά η χρήση μεγάλων σελίδων (περισσότερα στην ενότητα για τα THPs) [6].

2.1.3 Πολιτικές δέσμευσης μνήμης

Η διαδικασία δέσμευσης φυσικής μνήμης αποτελεί ένα από τα κύρια ζητήματα που απασχολούν το υποσύστημα εικονικής μνήμης και αφορά το πως θα διαχειριστούν τα υποσυστήματα την ζήτηση φυσικής μνήμης από τα προγράμματα. Υπάρχουν δύο προσεγγίσεις σε αυτό το ζήτημα με κύρια διαφορά τους το πότε δεσμεύεται φυσική μνήμη για τις ανάγκες ενός προγράμματος. Παρακάτω, αναλύονται οι δύο προσεγγίσεις και αναφέρονται τα μειονεκτήματα και πλεονεκτήματα της καθεμίας.

Eager Paging

Όταν ακολουθείται η τεχνική του eager paging, ο μηχανισμός διαχείρισης μνήμης δεσμεύει πλαίσια φυσικής μνήμης για μία διεργασία, την ίδια στιγμή που εκχωρεί αντίστοιχο εύρος στο εικονικό χώρο διευθύνσεων. Το πρόβλημα με αυτήν την τακτική είναι ότι σπαταλιέται φυσική μνήμη, διότι μπορεί η διεργασία να μην χρησιμοποιήσει πότε όλο το εύρος μνήμης που ζήτησε να δεσμεύσει [1].

Τα θετικά αυτής της προσέγγισης είναι μικρότερη χρονική επιβάρυνση κατά την πρώτη προσπέλαση σελίδων και μεγαλύτερο περιθώριο δημιουργίας μεγάλων συνεχόμενων αντιστοιχίσεων μεταξύ εικονικής και φυσικής μνήμης (translation contiguity) [2]. Όμως, για να υπάρξει αυτή η μεγάλη μεταφραστική συνέχεια απαιτείται να μην υφίσταται έντονος κατακερματισμός της μνήμης, ο οποίος είναι αναπόφευκτος σε συστήματα που τρέχουν για αρκετό χρόνο και δεν έχουν αποδοτικούς μηχανισμούς αντιμετώπισης του κατακερματισμού (π.χ. compaction), και για να εκμεταλλευτούν την συνέχεια αυτή σωστά τα σύγχρονα υπολογιστικά συστήματα χρειάζονται ειδικό υλικό, το οποίο δεν έχει υλοποιηθεί και τοποθετηθεί ακόμα σε εμπορική τεχνολογία.

Τα μειονεκτήματα της τεχνικής στην λειτουργία των σύγχρονων λειτουργικών συστημάτων είναι πολύ πιο σημαντικά. Μη γνωρίζοντας εξαρχής την συμπεριφορά ενός προγράμματος, ο μηχανισμός διαχείρισης μνήμης ακολουθώντας αυτήν την τακτική οδηγείται συχνά σε σπατάλη φυσικής μνήμης, διότι πολύ συχνά οι διεργασίες δεν χρησιμοποιούν όλον το εύρος μνήμης που ζήτησαν εξαρχής από τον μηχανισμό διαχείρισης μνήμης. Αυτό το πρόβλημα μπορεί να μην φαίνεται αρχικά αρκετά σημαντικό για μηχανήματα με πάρα πολύ διαθέσιμη φυσική μνήμη, όμως ακόμα και σε αυτά ένα κακογραμμένο πρόγραμμα μπορεί να δημιουργήσει μεγάλα προβλήματα, διότι μπορεί να στρεσάρει περιττώς την φυσική μνήμη και επόμενα προγράμματα που θα μπορούσαν να τρέξουν συγχρόνως δεν θα μπορούν να εκτελεστούν λόγω έλλειψης διαθέσιμης μνήμης. Λύση στα προβλήματα αυτά αποτέλεσε η στροφή των μηχανισμών εικονικής μνήμης στην τεχνική του demand paging, η οποία περιγράφεται στην συνέχεια.

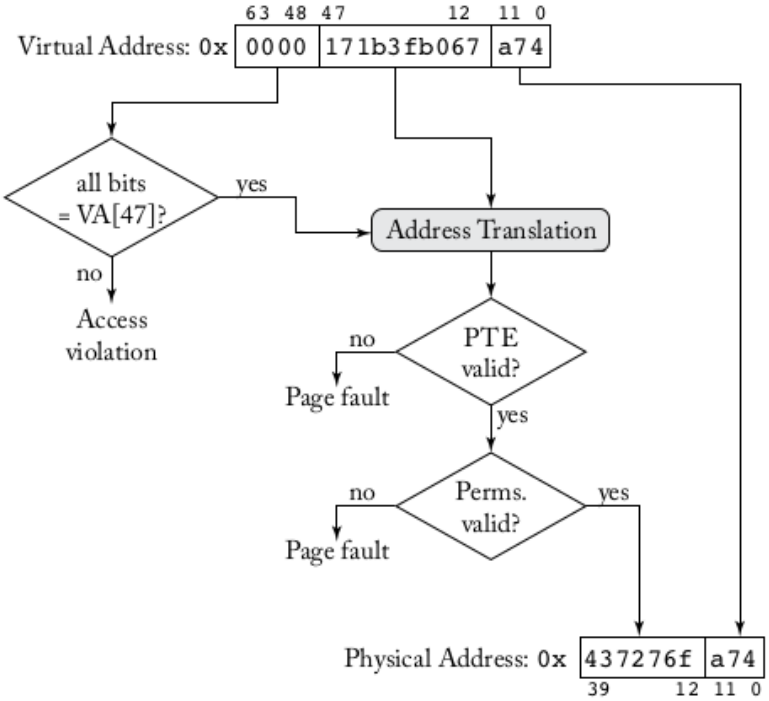
Demand Paging

Η τεχνική demand paging αποτελεί την κυρίαρχη προσέγγιση στην δέσμευση μνήμης στα σύγχρονα λειτουργικά συστήματα [1]. Όταν ο μηχανισμός δέσμευσης μνήμης ακολουθεί αυτήν την προσέγγιση, καθυστερεί την δέσμευση πλαισίων φυσικής μνήμης για τις εικονικές σελίδες ενός προγράμματος μέχρι να γίνει προσπάθεια πρόσβασης σε αυτά από το πρόγραμμα.

Αναλυτικότερα, όταν ένα πρόγραμμα ζητά χώρο στην μνήμη, επί της ουσίας το λ.σ. του εκχωρεί ένα συγκεκριμένο εύρος στον εικονικό χώρο διευθύνσεων και καθόλου στην φυσική μνήμη. Χώρος στην φυσική μνήμη για μία εικονική σελίδα δεσμεύεται την πρώτη φόρα που το πρόγραμμα θα προσπαθήσει να προσπελάσει την σελίδα, κατά την οποία προσπάθεια προκαλείται διακοπή (Page Fault) από τον επεξεργαστή, ο οποίος ¨καλεί¨ το λ.σ. να διαχειριστεί την διακοπή και να δεσμεύσει τον απαραίτητο φυσικό χώρο. Επομένως, στο τέλος το πρόγραμμα θα έχει δεσμεύσει τόσα φυσικά πλαίσια όσες και οι εικονικές σελίδες των οποίων τις διευθύνσεις χρησιμοποίησε.

Τα πλεονεκτήματα αυτής την τεχνικής είναι αρκετά. Πρώτον, δεσμεύεται τόση φυσική μνήμη όση πραγματικά θα χρειαστεί κάθε πρόγραμμα. Δεύτερον, ακόμα και αν έχει ζητηθεί αρκετή μνήμη από πολλά προγράμματα που τρέχουν συγχρόνως (αθροιστικά περισσότερη από την συνολική διαθέσιμη), μέχρι όντως να δεσμευτεί όλη η φυσική μνήμη θα μπορούν να εκτελούνται συγχρόνως δίχως προβλήματα. Τρίτον, υπάρχει λιγότερη χρονική επιβάρυνση κατά την αρχική ζήτηση μνήμης, η οποία επιβάρυνση ¨απλώνεται¨ κατά την εκτέλεση του προγράμματος και, επιπλέον, αποφεύγονται κοστοβόρες Ε/Ε διεργασίες (π.χ. μεταφορά δεδομένων από τον δίσκο στην μνήμη) για σελίδες που δεν είναι σίγουρο ότι θα χρησιμοποιηθούν.

Τα πλεονεκτήματα του demand paging είναι αναμφίβολα σημαντικά, για αυτό και είναι η κυρίαρχη πολιτική δέσμευσης μνήμης στα σύγχρονα λειτουργικά συστήματα. Όμως, στο demand paging δεν είναι όλα βέλτιστα. Πρώτον, εξαιτίας της αργοπορημένης και σταδιακής δέσμευσης φυσικών πλαισίων, δεν επιτυγχάνεται η υψηλή μεταφραστική συνέχεια που χαρακτηρίζει την λειτουργία του eager paging. Δεύτερον, αν πολλές εφαρμογές μαζί διεκδικούν ένα μικρό τελευταίο διαθέσιμο χώρο μνήμης, οδηγούμαστε στο φαινόμενο του thrashing, όπου οι υπολογιστικοί πόροι απασχολούνται υπερβολικά για την δέσμευση/αποδέσμευση σελίδων (swapping in/out) στην μνήμη ώστε να εξυπηρετήσουν όλα τα προγράμματα που τρέχουν στο σύστημα.



Σχήμα 2.1: *Διαδικασία μετάφρασης διευθύνσεων σε αρχιτεκτονική x86-64 [1]*

2.1.4 Μετάφραση διευθύνσεων

Σημαντικό τμήμα οποιουδήποτε μηχανισμού εικονικής μνήμης είναι η μετάφραση των διευθύνσεων. Η διαδικασία αυτή εκτελείται σε κάθε πρόσβαση στην μνήμη από ένα πρόγραμμα και έχει παρατηρηθεί σε προγράμματα με έντονη χρήση της μνήμης ότι έως και το 30-40%

των εντολών να αφορούν εντολές πρόσβασης στην μνήμη[1]. Επομένως, η μετάφραση διευθύνσεων επηρεάζει σημαντικά την επίδοση ενός προγράμματος και μπορεί να αποτελέσει bottleneck στην εκτέλεση του προγράμματος. Για την επιτάχυνση αυτής της διαδικασίας και την αποφυγή προβλημάτων στην επίδοση των προγραμμάτων, η μετάφραση διευθύνσεων γίνεται από ειδικούς μηχανισμούς στο υλικό σε συνδυασμό με εξεζητημένες λειτουργίες και διεργασίες από πλευράς λειτουργικού συστήματος.

Στην συνέχεια, θα αναφερθούμε σε μηχανισμούς υλικού και λογισμικού που σχεδιάστηκαν με γνώμονα την ταχύτερη μετάφραση των διευθύνσεων, δίνοντας ιδιαίτερη προσοχή σε αυτούς που υλοποιεί και χρησιμοποιεί το λειτουργικό σύστημα Linux με έμφαση στις υλοποιήσεις των μηχανισμών στην αρχιτεκτονική x86-64.

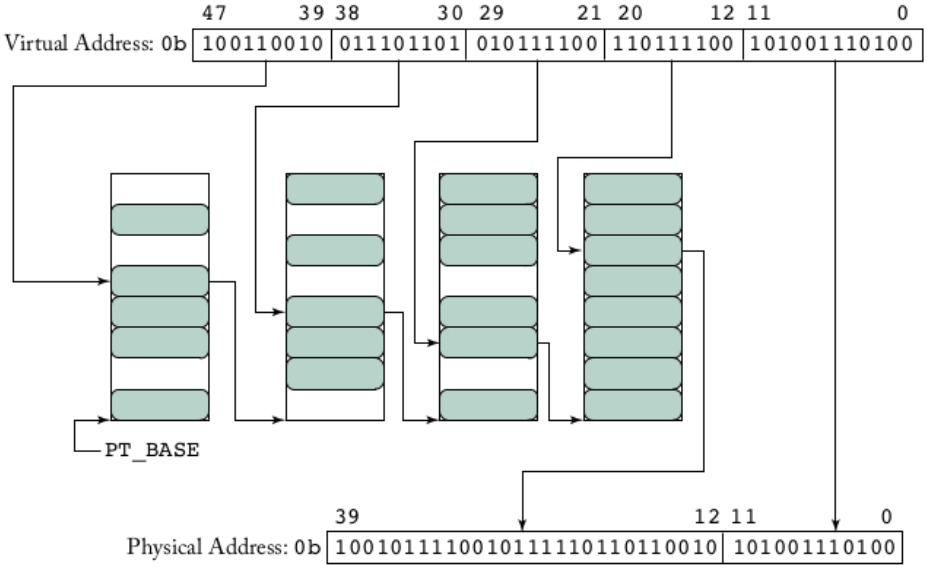
Πίνακας σελίδων

Ο πίνακας σελίδων (page table) είναι μία δομή δεδομένων που χρησιμοποιείται για την αποθήκευση των αντιστοιχίσεων μεταξύ φυσικών και εικονικών διευθύνσεων σε υποσυστήματα εικονικής μνήμης που κάνουν χρήση της τεχνικής της σελιδοποίησης. Κάθε διεργασία του συστήματος έχει το δικό της πίνακα σελίδων (ακόμα και ο πυρήνας), στον οποίο αποθηκεύονται μόνο πληροφορίες για την μετάφραση των αριθμών εικονικών σελίδων (VPN) στους αντίστοιχους αριθμούς φυσικών πλαισίων (PFN) που ανήκουν στην διεργασία. Πρόκειται για μία δομή δεδομένων κλειδιού-τιμής, όπου κλειδί είναι το VPN και τιμή το αντίστοιχο PFN. Οι καταχωρήσεις στον πίνακα σελίδων ονομάζονται page table entries (PTE) και κάθε τέτοια καταχώρηση περιέχει τον αριθμό φυσικού πλαισίου και μεταδεδομένα για την σελίδα στα υπόλοιπα bits (δικαίωμα εγγραφής ή και εκτέλεσης).

Ο πίνακας σελίδων πρέπει να καλύπτει το σύνολο του χώρου εικονικών διευθύνσεων, όπου το μέγεθος του τελευταίου δεν καθορίζεται από την διαθέσιμη φυσική μνήμη αλλά από το πλήθος των σελίδων που μπορούν να διευθυνσιοδοτηθούν βάσει της αρχιτεκτονικής του συστήματος. Συνεπώς, για να μην χαραμίζεται σημαντικός χώρος στην φυσική μνήμη (έως και 512GB για την αποθήκευση όλου του πίνακα στην αρχιτεκτονική x86-64), στις σύγχρονες υλοποιήσεις του μηχανισμού εικονικής μνήμης κυριαρχεί το σχέδιο του πολυεπίπεδου πίνακα σελίδων [1]. Ο πολυεπίπεδος πίνακας σελίδων είναι ένα ιεραρχημένο radix δέντρο, όπου ο αριθμός των επιπέδων εξαρτάται από την αρχιτεκτονική (π.χ. στην x86-64 έχει 4 επίπεδα) και το εύρος του χώρου διευθύνσεων. Οι πίνακες στα επίπεδα, εκτός του τελευταίου, αποτελούνται από δείκτες σε πίνακες του επόμενου επιπέδου (σχήμα 2.2). Στο τελευταίο επίπεδο οι πίνακες αποτελούνται από PTEs. Επίσης, σε περίπτωση που για ένα μεγάλο εύρος εικονικών διευθύνσεων δεν υπάρχουν δεσμευμένες σελίδες, το αντίστοιχο υποδέντρο παραμένει κενό, μέχρι να υπάρξει μία νέα καταχώρηση σε αυτό το εύρος. Οπότε, με αυτόν τον τρόπο παραμένει σχετικά μικρό το δέντρο, διότι συνήθως οι εφαρμογές δεν χρησιμοποιούν μεγάλο μέρος του εικονικού χώρου διευθύνσεων, και έτσι ο πολυεπίπεδος πίνακας σελίδων κλιμακώνεται αρμονικά με την αύξηση της χρήσης της φυσικής μνήμης από την διεργασία.

Ο πολυεπίπεδος πίνακας σελίδων πραγματοποιεί την μετάφραση σε βήματα, και η διαδικασία της διαδοχικής προσπέλασης των επιπέδων ενός πίνακα σελίδων ονομάζεται περπάτημα πίνακα σελίδων (page table walk). Αναλυτικότερα, η αρχική (προς μετάφραση) εικονική διεύθυνση διασπάται σε τμήματα και κάθε ένα χρησιμοποιείται σαν index στον κατάλληλο

πίνακα του αντίστοιχου επίπεδου. Η θέση στην οποία φτάνει το ¨περπάτημα¨ στο τελευταίο επίπεδο περιέχει, αν υπάρχει, την καταχώρηση με τον αριθμό του φυσικού πλαισίου που περιέχει τα δεδομένα της προς μετάφραση εικονικής διεύθυνσης.



Σχήμα 2.2: *Πολυεπίπεδος πίνακας σελίδων (x86-64) [1]*

Επίσης, ο σχεδιασμός των πολυεπίπεδων πινάκων σελίδων επιτρέπει την αρμονική συνύπαρξη σελίδων διαφορετικού μεγέθους (π.χ. 4KB σελίδα με μεγάλη σελίδα 2MB) [1]. Μία εγγραφή σε πίνακα ενδιάμεσου επιπέδου μπορεί να δείχνει κατευθείαν σε φυσική διεύθυνση αντί ενός πίνακα κατώτερου επιπέδου. Για παράδειγμα, στην αρχιτεκτονική x86-64 μία εγγραφή στο προτελευταίο επίπεδο μπορεί να δείχνει κατευθείαν σε μία μεγάλη σελίδα μεγέθους 2MB, δηλαδή μήκους 512 σελίδων βάσης (4KB), όσο και το μήκος ενός υποπίνακα στο τελευταίο επίπεδο.

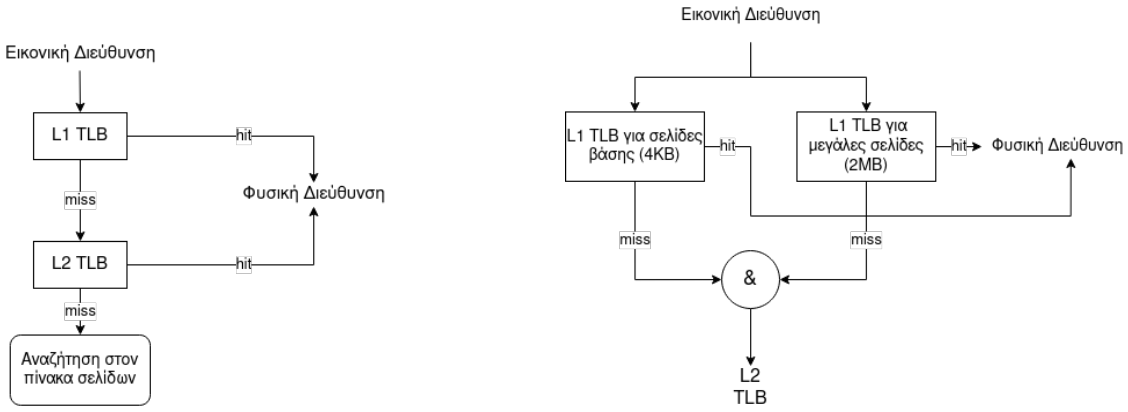
Translation Lookaside Buffer

Για την εξυπηρέτηση μιας εντολής προσπέλασης μνήμης ο επεξεργαστής πρέπει να μεταφράσει την εικονική διεύθυνση στην αντίστοιχη φυσική βάσει των πληροφοριών που είναι αποθηκευμένες στον πίνακα σελίδων. Η μετάφραση αυτή δεν είναι αμελητέου κόστους: μία προσπέλαση ενός πολυεπίπεδου πίνακα σελίδων (walking page table) απαιτεί αρκετές προσβάσεις στην φυσική μνήμη (4 στην αρχιτεκτονική του x86-64). Για λόγους επίδοσης, οι επεξεργαστές χρησιμοποιούν εξειδικευμένες κρυφές μνήμες (Translation Lookaside Buffers, TLBs), όπου αποθηκεύουν τις πιο πρόσφατες μεταφράσεις. Τα TLBs προσφέρουν τόσο μεγάλη βελτίωση στην επίδοση των υποσυστημάτων εικονικής μνήμης, ώστε δίχως αυτά η λειτουργία των υποσυστημάτων αυτών θα ήταν απαγορευτικά επιβαρυντική στην επίδοση των συστημάτων [1].

Η διατήρηση της ταχύτητας αυτών των κρυφών μνημών απαιτεί το μέγεθος τους να παραμένει μικρό. Όμως, όπως έχουμε ήδη αναφέρει ήδη, για να συνεχίζουν να προσφέρουν την πολύτιμη επιτάχυνση των μεταφράσεων για μία διεργασία, πρέπει να έχουν μία καλή κάλυψη της φυσικής μνήμης σε σχέση με την συνολικό αποτύπωμα της διεργασίας στην

μνήμη. Αυτό γίνεται όλο και δυσκολότερο όσο αυξάνονται οι απαιτήσεις σε πλήθος φυσικής μνήμης.

Τα τελευταία χρόνια έχουν υλοποιηθεί διάφορες λύσεις. Μία τέτοια λύση ήταν η υλοποίηση των μεγάλων σελίδων, που περιγράψαμε νωρίτερα, και η υποστήριξή τους από το TLB είτε με την δυνατότητα το TLB να εξυπηρετεί διαφορετικά μεγέθη σελίδων, είτε με την ύπαρξη ενός 2ου μικρότερου TLB αποκλειστικά για τις μεγαλύτερες σελίδες(Σχήμα 2.3β΄). Επιπλέον, για να αυξηθεί η κάλυψη και να μειωθούν οι προσπελάσεις του πίνακα σελίδων, μία άλλη σχεδίαση που έχει εφαρμοστεί είναι τα πολυεπίπεδα TLBs (Σχήμα 2.3α΄), ο οποίος σχεδιασμός είχε εφαρμοστεί επιτυχώς και στις λοιπές κρυφές μνήμες στους σύγχρονους επεξεργαστές.



(α΄) *TLB 2 επιπέδων*

(β΄) *Παράλληλα TLBs για διαφορετικά μεγέθη σελίδων*

Σχήμα 2.3: *Σχεδιασμοί για TLB*

2.2 Μηχανισμός διαχείρισης μνήμης στο Linux

Στο λειτουργικό σύστημα Linux, η διαχείριση της μνήμης αποτελεί ένα από τους σημαντικότερους και πιο πολύπλοκους μηχανισμούς του πυρήνα. Στην ενότητα αυτή, θα επικεντρωθούμε στην παρουσίαση κάποιων κύριων τμημάτων της διαχείρισης μνήμης, τα οποία είναι σημαντικά για την κατανόηση της διπλωματικής εργασίας. Τα τμήματα αυτά αποτελούν μηχανισμούς των οποίων η χρήση είναι δυνατή σε διάφορες αρχιτεκτονικές. Όμως εμείς σε τυχόν παραδείγματα που ακολουθούν θα εννοούμε την αρχιτεκτονική x86-64, γύρω από την οποία επικεντρώθηκε και η έρευνα που παρουσιάζεται σε επόμενη ενότητα.

Το λ.σ. Linux μπορεί να δεσμεύει, με ένα αίτημα, μνήμη μήκους μεγαλύτερου της μιας σελίδας. Συγκεκριμένα, πρέπει κάθε αίτημα να αφορά μήκος που είναι δύναμη του 2. Επιπλέον, για την καλύτερη επίδοση των μηχανισμών εικονικής μνήμης, δεν έχει απλά την δυνατότητα χρήσης μεγάλων σελίδων, αλλά με τον μηχανισμό των THPs δίνει την δυνατότητα καλύτερης εκμετάλλευσης των δυνατοτήτων των μεγάλων σελίδων. Παρακάτω, Θα περιγράψουμε αναλυτικά τους μηχανισμούς Buddy allocator, Transparent Huge Pages και θα αναφέρουμε κάποια μεταδεδομένα που είναι αναγκαία για την λειτουργία των μηχανισμών διαχείρισης μνήμης.

2.2.1 Μεταδεδομένα του μηχανισμού

Σε αυτήν την υποενότητα θα αναφερθούμε σε διάφορα μεταδεδομένα που διατηρεί ο μηχανισμός και στο πως αυτά αναπαρίστανται. Θα παρουσιαστούν κυρίως αυτά τα οποία αναφέρονται, στην συνέχεια της διπλωματικής εργασίας, σε διάφορους πειραματικούς μηχανισμούς και στην τελική υλοποίησή μας.

Περιγραφητής μνήμης (Memory Descriptor)

Ο μηχανισμός διαχείρισης μνήμης του λ.σ. βασίζεται στην διατήρηση κάποιων μεταδεδομένων. Κάθε διεργασία έχει το δικό της αντικείμενο του τύπου δεδομένων *struct mm_struct* [7], το οποίο περιέχει όλες τις πληροφορίες για τον χώρο διευθύνσεων της σχετικής διεργασίας.
Παρακάτω αναφέρονται τα σημαντικότερα πεδία που περιέχει η δομή αυτή:

- Πληροφορίες για τις διευθύνσεις των διάφορων τμημάτων της διεργασίας (π.χ. εκτελέσιμος κώδικας, αρχικά δεδομένα, σωρός, στοίβα) και κατάλληλους μετρητές, όπως για τις διάφορα τύπου σελίδων που ανήκουν στο mm_struct ή για το πλήθος διεργασιών που μοιράζονται τον ίδιο περιγραφητή.
- Αναφορά στον πίνακα σελίδων της διεργασίας και συγκεκριμένα δείκτη προς τον αντίστοιχο καθολικό κατάλογο σελίδων (Page Global Directory).
- Μία λίστα με τα εύρη εικονικής μνήμης (memory regions) που έχουν εκχωρηθεί στην διεργασία.
- Για γρηγορότερη αναζήτηση στην λίστα, υπάρχει και ένα δέντρο, στο οποίο βρίσκονται τα στοιχεία της λίστας ταξινομημένα κατά εικονική διεύθυνση.

- Semaphore για την ανάγνωση/τροποποίηση των πληροφοριών του mm_struct.

Εύρος εικονικής μνήμης (Memory Region)

Κάθε εύρος εικονικής μνήμης (VMA) που έχει ο περιγραφητής μνήμης μίας διεργασίας αναπαρίσταται από ένα αντικείμενο του τύπου δεδομένων *struct vm_area_struct* [7]. Κάθε τέτοιο αντικείμενο περιέχει τα αναγκαία προσδιοριστικά του αντίστοιχου εύρους μνήμης. Αυτά είναι τα όρια του εύρους, τα δικαιώματα πρόσβασης στις σελίδες του εύρους και διάφορα κατάλληλα πεδία ανάλογα με το αν πρόκειται για VMA με ανώνυμες σελίδες ή σελίδες υποστηριζόμενες από αρχεία.

Περιγραφητής σελίδας (Page descriptor)

Για τις ανάγκες του πυρήνα, διατηρούνται πληροφορίες για όλα τα φυσικά πλαίσια που διαχειρίζεται. Κάθε φυσικό πλαίσιο έχει το δικό του περιγραφητή που είναι αντικείμενο του τύπου δεδομένων **struct page**. Το αντικείμενο αυτό περιέχει κατάλληλες πληροφορίες για την κατάσταση της σελίδας (μέσω flags), τον αριθμό των αναφορών (πλήθος PTEs με αυτό το φυσικό πλαίσιο) και λοιπά πεδία χρήσιμα για διάφορους μηχανισμούς του λ.σ. (π.χ. buddy allocator, PageCache).

Ζώνη μνήμης (Memory zone)

Οι σύγχρονες αρχιτεκτονικές δημιουργούν περιορισμούς στο πως τα φυσικά πλαίσια μπορούν να χρησιμοποιηθούν. Για παράδειγμα, κάποιες συσκευές μπορούν να έχουν άμεση πρόσβαση μόνο σε περιορισμένο εύρος μνήμης (π.χ. στα πρώτα 16MB). Αυτοί οι περιορισμοί δημιούργησαν την ανάγκη για χωρισμό της φυσικής μνήμης σε κάποιες ζώνες [7]. Συγκεκριμένα στην αρχιτεκτονική x86-64, στα συστήματα 64bit υπάρχουν κυρίως 3 ζώνες (ZONE_DMA, ZONE_DMA32, ZONE_NORMAL), όπου η τελευταία χρησιμοποιείται κυρίως για την παροχή σελίδων για τις ανάγκες των διεργασιών επιπέδου χρήστη (user-space).

Κάθε ζώνη περιγράφεται από ένα αντικείμενο του τύπου δεδομένων *struct zone*. Το αντικείμενο αυτό περιέχει πληροφορίες για την κατάσταση της ζώνης (π.χ. ελεύθερες σελίδες και διάφοροι μετρητές για τις σελίδες που ανήκουν στην ζώνη). Επίσης, περιέχει τις λίστες με τα ελεύθερα blocks μνήμης της ζώνης , τις οποίες λίστες χρησιμοποιεί ο buddy allocator (θα αναφερθούμε στην συνέχεια σε αυτό).

2.2.2 Buddy Allocator

Ο buddy allocator αποτελεί τον βασικό μηχανισμό εντοπισμού διαθέσιμων σελίδων για δέσμευση τους από το λ.σ. Linux. Η τεχνική αυτή σχεδιάστηκε για να αποφύγει τον εξωτερικό κατακερματισμό (external fragmentation). Αυτό το πετυχαίνει, καταγράφοντας τα διαθέσιμα υπάρχοντα blocks συνεχόμενων ελεύθερων πλαισίων μνήμης, ώστε να αποφεύγεται η ανάγκη για διάσπαση ενός μεγάλου block μνήμης με σκοπό την ικανοποίηση μίας ανάγκης για μικρότερο χώρο [7].

Ο μηχανισμός ομαδοποιεί τα ελεύθερα πλαίσια σε 11 λίστες, όπου κάθε λίστα περιέχει τα blocks συνεχόμενης ελεύθερης μνήμης συγκεκριμένου μήκους. Συγκεκριμένα, η i-οστή

λίστα περιέχει τα blocks μήκους 2^i (η αρίθμηση ξεκινάει από το 0), οπότε το καθιερωμένο μεγαλύτερο μήκος block που μπορεί να καταγράψει ο πυρήνας είναι 1024 σελίδων βάσης (2MB). Επιπλέον, αν και αυτό το πλήθος λιστών έχει καθιερωθεί εδώ και χρόνια, υπάρχει σχετική έρευνα με πειραματισμούς με μεγαλύτερο πλήθος λιστών όπου τα αποτελέσματα έδειξαν σε συγκεκριμένες περιπτώσεις-υλοποιήσεις σχετική βελτίωση σε σχέση με τον κλασσικό αριθμό λιστών [4, 2].

Κατά την εκκίνηση του λ.σ., αρχικοποιούνται οι λίστες χωρίζοντας την φυσική μνήμη στα μεγαλύτερα δυνατά blocks που αναγνωρίζει το λ.σ. και είναι αναμενόμενο ότι αρχικά όλη η φυσική μνήμη θα αποτυπώνεται κυρίως σε blocks μέγιστου μεγέθους. Μετά την αρχικοποίηση των λιστών, ο μηχανισμός είναι έτοιμος να καλύψει τα αιτήματα για μνήμη από την λίστα με το μικρότερο δυνατό μήκος στα blocks.
Αναλυτικά ο αλγόριθμος επιλογής:

1. Αν υπάρχει διαθέσιμο block στην λίστα με block ίδιου μήκους με το αίτημα,τότε αφαιρείται το block από την λίστα και ικανοποιείται το αίτημα.

2. Αλλιώς, διατρέχει τις λίστες βηματικά προς τα πάνω (λίστες με blocks μεγαλύτερου μήκους) μέχρι να βρει λίστα με διαθέσιμο block ικανό να ικανοποιήσει το αίτημα. Αν φτάσει στην ανώτερη λίστα (λίστα μέγιστης τάξης) και δεν βρει διαθέσιμο block. Το αίτημα αποτυγχάνει, λόγω μη αρκετά διαθέσιμης μνήμης.

3. Αν βρει διαθέσιμο block, τότε επειδή είναι μεγαλύτερο αυτό, πρέπει να το διασπάσει ώστε να πάρει ένα τμήμα του που να είναι κατάλληλου μεγέθους. Αυτό γίνεται μέσα από μία επαναληπτική διάσπαση του διαθέσιμου block, όπου τα αχρησιμοποίητα τμήματα του αρχικού μεγάλου block, εισάγονται στις αντίστοιχες λίστες του buddy allocator και το τελικό κατάλληλο block δεσμεύεται και το αίτημα ικανοποιείται.

Θα το εξηγήσουμε με ένα απλό παράδειγμα αιτήματος δέσμευσης μνήμης. Έστω ότι όλα τα διαθέσιμα blocks βρίσκονται στην μέγιστου μήκους λίστα (blocks μήκους 1024 πλαισίων) και το αίτημα αφορά ένα block 256 πλαισίων.
Τα βήματα που κάνει ο αλγόριθμος είναι τα εξής:

1. Ελέγχει αν υπάρχει διαθέσιμο block στην λίστα με τα block μήκους 256, την οποία βρίσκει κενή.

2. Προχωράει στην προσπέλαση της επόμενης λίστας με blocks μήκους 512, την οποία επίσης βρίσκει κενή. Οπότε, προχωράει στην επόμενη και τελευταία λίστα των blocks 1024 σελίδων.

3. Αφού η τελευταία λίστα είναι μη κενή, παίρνει το πρώτο διαθέσιμο block στην λίστα.

4. Το ζητούμενο block είναι μικρότερο από το επιλεγμένο. Οπότε, ξεκινά η διαδικασία διάσπασής του (χωρισμός σε 2 ισομερή blocks).

5. Η πρώτη διάσπαση δημιουργεί 2 blocks 512 σελίδων, τα οποία συνεχίζουν να είναι μεγαλύτερα από το ζητούμενο, οπότε επιλέγει για το επόμενο βήμα το πρώτο block και τοποθετεί το δεύτερο block στην σχετική λίστα του buddy allocator.

6. Το επιλεγμένο block διασπάται ξανά και δημιουργεί 2 νέα blocks 256 σελίδων, τα οποία είναι του επιθυμητού μήκους. Συνεπώς, επιλέγει το ένα για να δεσμευτεί για το αίτημα και το δεύτερο το τοποθετεί πίσω στην κατάλληλη λίστα.

2.2.3 Transparent Huge Pages (THPs)

Προβλήματα στην χρήση μεγάλων σελίδων

Η χρήση μίας μεγάλης σελίδας ως μονάδα στους μηχανισμούς εικονικής μνήμης έχει σημαντικά μειονεκτήματα [8]. Πρώτον, η χρήση μεγάλων σελίδων σε ένα υπολογιστικό σύστημα αυξάνει την εμφάνιση εσωτερικού κατακερματισμού. Δεύτερον, αν πραγματοποιείται αραιή πρόσβαση στην μνήμη (sparsely-accessed memory), η διαχείριση των μεγάλων σελίδων είναι αρκετά κοστοβόρα και ασύμφορη σε σχέση με την επιθυμητή βελτίωση της απόδοσης του συστήματος. Αυτό προκύπτει από το γεγονός ότι η εξυπηρέτηση ενός αιτήματος για μεγάλη σελίδα προκαλεί αρκετά μεγαλύτερη χρονική επιβάρυνση στην διαδικασία εκχώρησης φυσικής μνήμης σε σχέση με ένα αίτημα για σελίδα βάσης (π.χ. 4KB) (π.χ. λειτουργίες Ε/Ε και καθαρισμός νέας σελίδας). Αυτά τα ζητήματα όχι μόνο έχουν εμποδίσει την καθιέρωση των μεγάλων σελίδων, αλλά σε ορισμένα σενάρια παρατηρείται χειρότερη επίδοση όταν χρησιμοποιούνται, [6], και για αυτό κάποια συστήματα βάσεων δεδομένων συνιστούν την μη χρήση τους.

Hugetlbfs

Η πρώτη ενσωμάτωση των μεγάλων σελίδων στο λ.σ. Linux για χρήση από προγράμματα (user-space) έγινε με τον μηχανισμό hugetlbfs. Όμως, το hugetlbfs είναι μία πολύ περιοριστική προσέγγιση, η οποία απευθύνεται σε συγκεκριμένα σενάρια και σε προγραμματιστές που επιθυμούν ρητά την χρήση μεγάλων σελίδων για τα προγράμματά τους, ενώ δεν επιτρέπει την μικτή χρήση μεγάλων και μικρών σελίδων στο ίδιο mapping.

Η λύση μέσω των THPs

Ο σχεδιασμός του μηχανισμού των διάφανων μεγάλων σελίδων, (Transparent Huge Pages) αποτέλεσε προϊόν της προσπάθειας να αφαιρεθεί η αποσύνδεση μεταξύ μεγάλων σελίδων και του βασικού μηχανισμού διαχείρισης μνήμης του Linux [8]. Η εισαγωγή των THPs στον βασικό μηχανισμό διαχείρισης μνήμης του λ.σ. Linux προσέφερε τις εξής νέες δυνατότητες:

- Με τα THPs, ο μηχανισμός διαχείρισης μνήμης μπορεί να διασπάσει μία μεγάλη σελίδα, όπου χρειάζεται, στις σελίδες βάσης που την απαρτίζουν εύκολα και αποδοτικά. Αυτό είναι ιδιαίτερα χρήσιμο για λειτουργίες που μετακινούν σελίδες (π.χ. swap, migrations), οι οποίες δουλεύουν κυρίως με σελίδες βάσης.

- Πέρα από την διάσπαση, το λ.σ. μπορεί και να ενώσει σελίδες που βρίσκονται σε συνεχόμενα πλαίσια βάσης σε μία μεγάλη σελίδα, όπου οι μεμονωμένες σελίδες είναι κατάλληλα ευθυγραμμισμένες. Αυτή η δυνατότητα σε συνδυασμό με μετακινήσεις σελίδων βάσης επιτρέπει την συγκέντρωση από διάφορα σημεία σελίδων σε κατάλληλες

συνεχόμενες θέσεις, ώστε να ενωθούν σε μία μεγάλη σελίδα (η λειτουργία του δαίμονα khugepaged).

- Ο προγραμματιστής έχει μεγαλύτερη ευελιξία στην χρήση μεγάλων σελίδων. Αρκεί να δηλώσει για ένα mapping ότι επιθυμεί να υποστηριχτεί από μεγάλες σελίδες (με κλήση της madvise()). Αν δεν υπάρχει διαθέσιμος χώρος για μεγάλες σελίδες, η αναζήτηση χώρου δεν αποτυγχάνει αλλά επιστρέφει στην αναζήτηση μεμονωμένων σελίδων βάσης. Επίσης, μπορεί να δοθεί εντολή στο λ.σ. να προσπαθεί πάντα, πρώτα, για μεγάλες σελίδες όπου αυτό είναι εφικτό.

2.3 Πειραματικές επεκτάσεις μηχανισμών διαχείρισης μνήμης

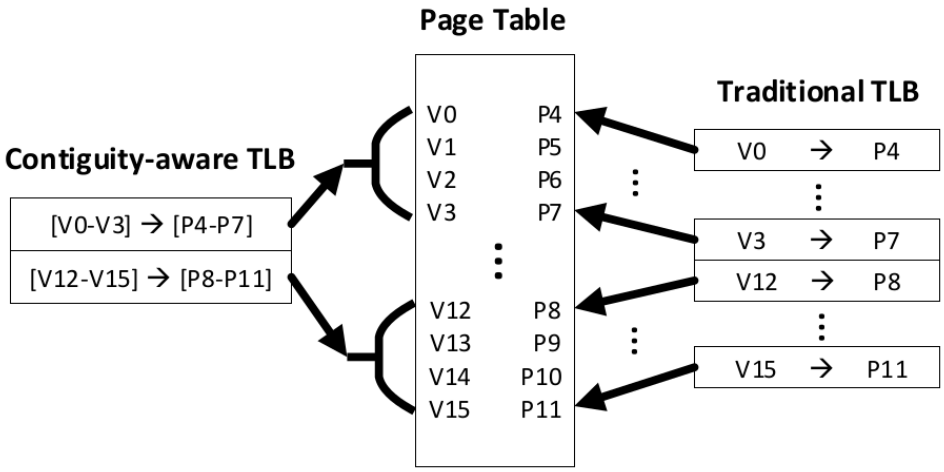
Στην ενότητα αυτή θα παρουσιαστούν πειραματικοί μηχανισμοί, που αποσκοπούν στην περαιτέρω βελτίωση της μετάφρασης διευθύνσεων. Επικεντρώνονται, κυρίως, στο ζήτημα της περιορισμένης κάλυψης των υπαρχόντων TLBs (limited TLB reach problem), το οποίο οδηγεί σε πολλά αποτυχίες στο TLB, οι οποίες με την σειρά τους εξαναγκάζουν την προσπέλαση του πίνακα σελίδων από το σύστημα, το οποίο προκαλεί επιπλέον αποτυχίες σε αναζητήσεις στην κρυφή μνήμη.

Οι παρακάτω μηχανισμοί δεν έχουν ενσωματωθεί σε εμπορικό προϊόν, αλλά αποτελούν κυρίως σχεδιαστικές προτάσεις. Όμως, έχουν αξιολογηθεί μέσω πειραμάτων και ειδικών προσομοιώσεων σε σύγχρονα υπολογιστικά συστήματα με αρκετά καλά αποτελέσματα και αποτελούν προτάσεις ικανές να μετριάσουν ή και να λύσουν το πρόβλημα της περιορισμένης κάλυψης από τα υπάρχοντα TLB.

2.3.1 Redundant Memory Mappings

Εκτελώντας διάφορα πειράματα παρατηρήθηκε ότι στις περισσότερες περιπτώσεις εμφανιζόταν μία μεγάλη συνέχεια στον εικονικό χώρο διευθύνσεων και ο αριθμός των ευρών που χρειάζονταν για την 99% κάλυψή της ήταν λιγότερο από 50 [2]. Αυτό ενέπνευσε τον σχεδιασμό του μηχανισμού **Redundant Memory Mappings** (RMM), ο οποίος αποτελεί μία νέα ολοκληρωμένη (σε επίπεδο υλικού και λογισμικού) επέκταση του μηχανισμού εικονικής μνήμης. Ο μηχανισμός RMM διατηρεί τον υπάρχοντα μηχανισμό *demand paging* του λειτουργικού συστήματος και προσθέτει την δημιουργία και αποθήκευση πλεονάζοντων/εφεδρικών συσχετίσεων (redundant mapping), η οποία επιτρέπει την αποδοτικότερη εκμετάλλευση της μεταφραστικής συνέχειας μεγάλων συνεχών αντιστοιχίσεων σελίδας.

Ο μηχανισμός βασίζεται στην ιδέα της μονής μετάφρασης ενός ολόκληρου αυθαίρετα μεγάλου εύρους διευθύνσεων (range translation). Κάθε τέτοια αντιστοίχιση ενός αυθαίρετου συνεχούς εύρους εικονικών διευθύνσεων με συνεχόμενες φυσικές σελίδες χαρακτηρίζεται από μία τριάδα αριθμών (BASE, LIMIT και OFFSET). Το BASE αφορά την αρχή του εικονικού εύρους (virtual page number), το LIMIT καθορίζει το τέλος του εύρους στον εικονικό χώρο διευθύνσεων και το OFFSET είναι η απόσταση (διαφορά) μεταξύ των αρχικών διευθύνσεων στον φυσικό και εικονικό χώρο ($OFFSET = αρχικPFN - αρχικVPN$). Ακολουθεί η αναλυτική παρουσίαση του μηχανισμού στα πλαίσια της αρχιτεκτονικής (x86-64), με την οποία ασχολούμαστε κυρίως στην παρούσα διπλωματική εργασία.



Σχήμα 2.4: *Σύγκριση κάλυψης ενός range TLB με τον κλασσικό TLB σελίδων*

Range TLB

Το αναγκαίο υλικό για τον μηχανισμό RMM είναι κυρίως ο range TLB που διατηρεί πολλαπλές μεταφράσεις συνεχών αντιστοιχίσεων (range translations). Πρόκειται για μία πλήρως συσχετιστική κρυφή μνήμη (fully-associative cache), διότι κάθε καταχώρηση σε αυτήν μπορεί να είναι οποιουδήποτε μεγέθους. Κάθε καταχώρηση στο range TLB αποτελείται από τα BASE, LIMIT και OFFSET, τα οποία έχουν αναφερθεί παραπάνω. Επίσης, μέσα στο OFFSET διατηρούνται και τα bits προστασίας (protection bits) του εύρους.

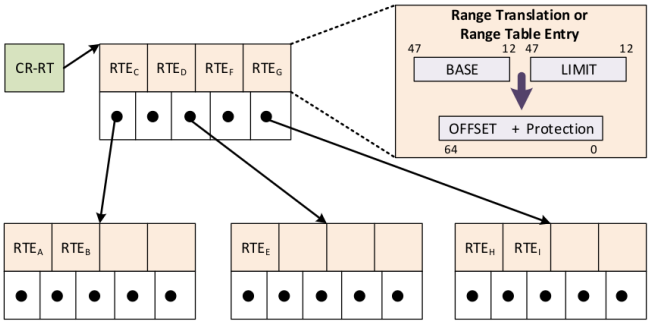
Η σχεδιαστική επιλογή που αναφέρεται από τους δημιουργούς του μηχανισμού είναι η πρόσβαση στο range TLB να γίνεται παράλληλα με το L2 TLB, μετά από αποτυχία στο L1 TLB (Σχήμα 2.6). Ο range TLB ψάχνει για το VPN που απέτυχε να μεταφραστεί στο L1 TLB και σε περίπτωση επιτυχημένης αναζήτησης, επιστρέφει το αντίστοιχο PFN. Σε περίπτωση αποτυχία στο range TLB, ακολουθεί αναζήτηση στους πίνακες ευρών (range tables) και, αν υπάρχει, φέρνει την αντίστοιχη καταχώρηση. Επίσης, επειδή το range TLB προσπελάζεται παράλληλα με το TLB σελίδων του τελευταίου επιπέδου, περιέχει αναγκαστικά λίγες καταχωρήσεις ώστε να καταφέρνει να επιστρέφει αποτέλεσμα πριν το TLB σελίδων.

Range table

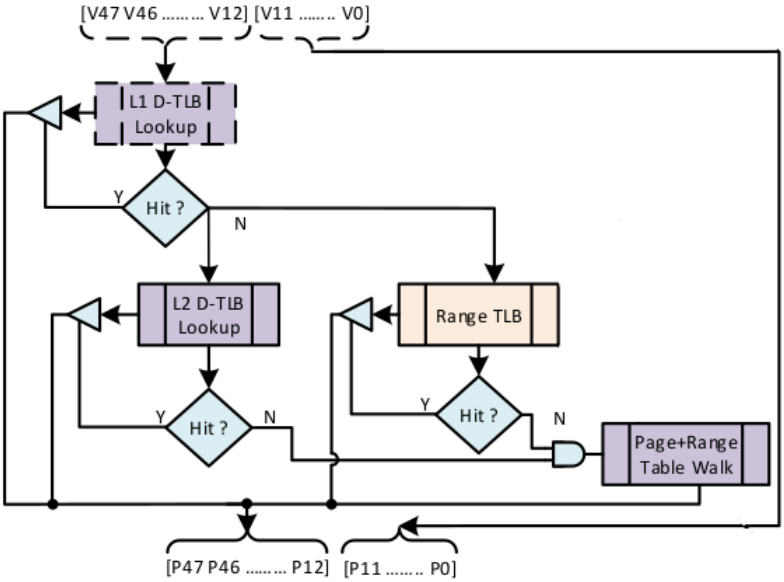
Ο πίνακας ευρών (range table) είναι μία δομή δεδομένων παρόμοια με αυτήν του πίνακα σελίδων. Κάθε διεργασία έχει τον δικό της πίνακα ευρών και αποθηκεύει σε αυτόν τις μεταφράσεις συνεχών αντιστοιχίσεων (range translations) της εφαρμογής. Επίσης, η υλοποίηση συνοδεύεται από κατάλληλο περιπατητή (walker) παρόμοιο με αυτόν για τους πίνακες σελίδων, ενώ η διαχείριση των καταχωρήσεων στον πίνακα γίνεται από το λειτουργικό σύστημα (θα αναλυθεί στην συνέχεια αυτό το τμήμα του μηχανισμού).

Για την υλοποίηση του πίνακα ευρών προτείνεται η χρήση B-Trees με κλειδί το ζεύγος (BASE, LIMIT) και με τιμή το πεδίο OFFSET (Σχήμα 2.5). Η επιλογή αυτής της δομής δεδομένων έγινε επειδή είναι φιλική προς τις caches και διατηρεί τα δεδομένα ταξινομημένα ώστε η αναζήτηση και ανανέωση των καταχωρήσεων να παίρνει λογαριθμικό χρόνο. Επίσης, συγκριτικά με έναν πίνακα σελίδων, ο πίνακας ευρών δεσμεύει αρκετά λιγότερο αποθηκευ-

τικό χώρο, διότι μία σελίδα 4KB αρκεί για να αποθηκεύσει έναν πίνακα ευρών (με χρήση B-tree) αρκετά μεγάλο για 128 καταχωρήσεις.



Σχήμα 2.5: *Ο πίνακας ευρών (range table) [2]*



Σχήμα 2.6: *Αρχιτεκτονική διάταξη της μετάφρασης διευθύνσεων με χρήση και range TLB [2]*

Υποστήριξη στο λειτουργικό σύστημα

Σημαντικό τμήμα του μηχανισμού RMM αποτελεί η επέκταση του μηχανισμού διαχείρισης φυσικής μνήμης του λειτουργικού συστήματος. Όπως αναφέρθηκε, το λειτουργικό σύστημα είναι υπεύθυνο για την διαχείριση του πίνακα ευρών, την αρχικοποίησή του και την δημιουργία καταχωρήσεων σε αυτό.

Η ανανέωση του πίνακα γίνεται κατά την εκτέλεση του βασικού μηχανισμού διαχείρισης μνήμης, δηλαδή κατά την δέσμευση, απελευθέρωση ή ανάκτηση σελίδων από το λειτουργικό σύστημα. Το λ.σ. ,για να αποφύγει την δημιουργία καταχωρήσεων για αρκετά μικρά εύρη, ενημερώνει τον πίνακα ευρών μόνο όταν πρόκειται για συνέχεια μεγαλύτερή από έναν αριθμό σελίδων. Το όριο αυτό μεταβάλλεται δυναμικά από το λ.σ., ανάλογα με την τρέχουσα εικόνα (πλήθος και μέγεθος καταχωρήσεων) του πίνακα ευρών.

Το λειτουργικό σύστημα, με το βασικό μηχανισμό εκχώρησης μνήμης (buddy allocator),

δεν μπορεί να πετύχει την δημιουργία λίγων μεταφράσεων για αρκετά μεγάλες συνεχόμενες αντιστοιχίσεις. Για αυτό το λόγο, ο μηχανισμός RMM τροποποιεί το βασικό μηχανισμό εκχώρησης μνήμης ώστε να χρησιμοποιεί την τεχνική του *eager paging*.Σε αντίθεση με την τεχνική του demand paging όπου τα φυσικά πλαίσια δεσμεύονται κατά την πρώτη πρόσβαση της διεργασίας στο εύρος μία εικονικής σελίδας, στο eager paging τα πλαίσια μνήμης δεσμεύονται κατά την εκχώρηση εύρους μνήμης στην διεργασία (π.χ. κατά την κλήση της mmap() ή της brk()). Με αυτόν τον τρόπο, πραγματοποιείται μία μαζική δέσμευση μνήμης, η οποία οδηγεί στην δημιουργία λίγων μεταφράσεων που αφορούν μεγαλύτερες (σε μήκος) συνεχόμενες αντιστοιχίσεις.

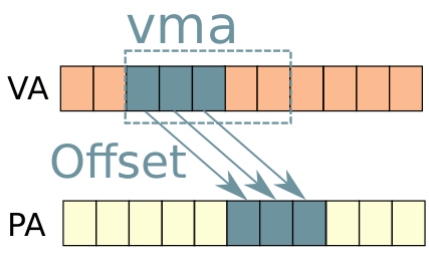
2.3.2 Contiguity-aware Paging

Ο Contiguity-aware Paging (CaP ή CaPaging), [3], είναι ένας νέος ευέλικτος μηχανισμός δημιουργίας αντιστοιχίσεων πολλών συνεχόμενων σελίδων σε επίπεδο λογισμικού. Αποτελεί επέκταση του μηχανισμού δέσμευσης μνήμης και λειτουργεί κατά την δέσμευση φυσικής μνήμης (εξυπηρέτηση των page fault), όπου τροποποιώντας τον μηχανισμό επιλογής πλαισίου, διαλέγει την κατάλληλη θέση με σκοπό την επίτευξη υψηλής μεταφραστικής συνέχειας.

Βασικά Χαρακτηριστικά

Ο CaPaging δημιουργεί εύρη συνεχόμενων αντιστοιχίσεων (contiguous mappings), όπου κάθε εύρος έχει το δικό του Offset. Αυτό το Offset αποτελεί την διαφορά *εικονικδιεθυνση – φυσικδιεθυνση* για μία αντιστοίχιση για μεγάλη σελίδα ή σελίδα βάσης και πρέπει όλες οι (υπάρχουσες) αντιστοιχίσεις ενός εύρους να έχουν το ίδιο Offset. Σκοπός του CaPaging είναι να δημιουργεί όσο το δυνατόν μεγαλύτερα τέτοια εύρη βασιζόμενος στο Offset που έχει προεπιλέξει. Η επιλογή του Offset σε κάθε περίπτωση δεν γίνεται αυθαίρετα αλλά βάσεις της τρέχουσας εικόνας της φυσικής μνήμης, η οποία αποτυπώνεται στο contiguity map (θα αναφερθούμε πιο κάτω σε αυτό).

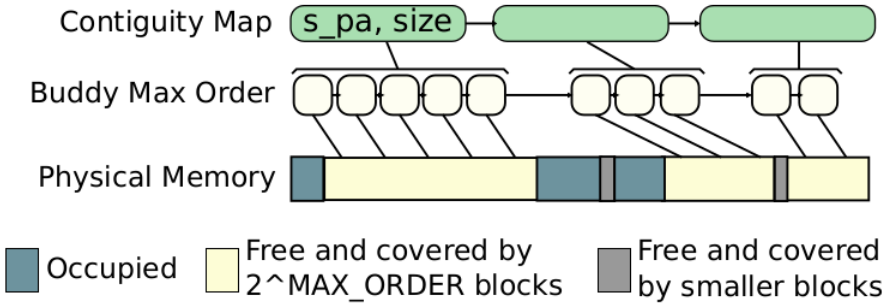
Η ευελιξία του μηχανισμού προκύπτει από το γεγονός ότι, αν και δεν διαλέγει τυχαία σελίδες αλλά επιδιώκει την τοποθέτηση των σελίδων σε συγκεκριμένες θέσεις, ο CaPaging συνδέεται αρμονικά με τον υπάρχοντα μηχανισμό του demand paging στα σύγχρονα λειτουργικά συστήματα. Δεν προδεσμεύει χώρο και δεν αλλάζει το πότε θα γίνει η δέσμευση των σελίδων. Η μόνη αλλαγή στην διαδικασία δέσμευσης μνήμης είναι, κατά την εξυπηρέτηση των page faults, να προσπαθεί αρχικά ο CaPaging να διαλέξει κατάλληλη θέση με μικρή χρονική επιβάρυνση και αν αποτύχει να αναλαμβάνει έπειτα ο buddy allocator.



Σχήμα 2.7: *Εύρος συνεχόμενων αντιστοιχίσεων (contiguous mapping) [3]*

Contiguity Map

Ο **χάρτης συνέχειας** (contiguity map) αποτελεί έναν χάρτη της φυσικής μνήμης, που αποτυπώνει τα διαθέσιμα συνεχόμενα εύρη μνήμης. Πρόκειται για μία λίστα από στοιχεία, όπου το κάθε στοιχείο είναι ένα διακριτό εύρος στην φυσική μνήμη. Για να μην γεμίσει με πολλά μικρού μήκους ranges και αυξηθεί το κόστος αναζήτησης σημαντικά, υπάρχει κατώτερο όριο μήκους. Το όριο είναι το μέγιστου μεγέθους block ελεύθερης φυσικής μνήμης που μπορεί να καταγράψει ο buddy allocator.



Σχήμα 2.8: *Ο χάρτης συνέχειας (contiguity map) [3]*

Για την αποδοτική λειτουργία του contiguity map και για να γίνεται γρήγορα η συσχέτιση ενός block μέγιστης τάξης του buddy allocator με το αντίστοιχο εύρος του στον χάρτη, τα ελεύθερα block μέγιστης τάξης έχουν ένα δείκτη προς το εύρος στο χάρτη που τους αντιστοιχεί. Για αυτό την σύνδεση κάνουν χρήση του πεδίου *mapping* στο struct page το οποίο μένει αχρησιμοποίητο για τις σελίδες των blocks που ανήκουν στις buddy λίστες.

Τα στοιχεία του χάρτη (εύρη) μεταβάλλονται μόνο όταν δημιουργούνται και καταστρέφονται στοιχεία της μέγιστης-τάξης λίστας ελεύθερων blocks του buddy allocator, η οποία διατηρείται ταξινομημένη για την αποφυγή κατακερματισμού ευρών μεγάλων συνεχόμενων ελεύθερων πλαισίων. Όταν, αφαιρείται ένα block από την λίστα μέγιστης τάξης του buddy allocator, γίνεται μη διαθέσιμο το εύρος του στο contiguity map. Αν βρίσκεται στα άκρα ενός μεγαλύτερου εύρους, συμπυκνώνεται το υπάρχον, ενώ όταν βρίσκεται εσωτερικά, διασπάται το παλιό εύρος σε δύο νέα δίχως το μη διαθέσιμο block.Όταν μετά από αποδεσμεύσεις σελίδων δημιουργείται ένα block μνήμης μέγιστης τάξης (MAX_ORDER στις free lists), αφού τοποθετηθεί στην μέγιστη λίστα, ενημερώνεται και ο χάρτης, όπου το νέο εύρος ενώνεται με κάποιο υπάρχον γειτονικό του ή μπαίνει ως ξεχωριστό εύρος στον χάρτη.

Η αναζήτηση στον χάρτη γίνεται μέσω ενός δείκτη που διατρέχει όλα τα στοιχεία της λίστας ανεξάρτητα από το σημείο εκκίνησης της αναζήτησης. Αν ξεκινήσει από ενδιάμεσο στοιχείο, όταν φτάσει στο τέλος ξεκινάει ξανά από την αρχή μέχρι το στοιχείο εκκίνησης. Η αναζήτηση τερματίζει πρόωρα αν βρει αρκετά μεγάλο διαθέσιμο χώρο. Αν δεν υπάρχει τέτοιο εύρος, διατρέχει όλα τα στοιχεία και η τελική επιλογή είναι το μεγαλύτερο εύρος που βρέθηκε. Τέλος, ο δείκτης μετακινείται ώστε να δείχνει στο τέλος του εύρους που επιλέχθηκε για το ζητούμενο χώρο. Αυτό γίνεται, ώστε σε επόμενη αναζήτηση να ελέγχεται τελευταίο το εύρος που επιλέχθηκε στην προηγούμενη αναζήτηση.

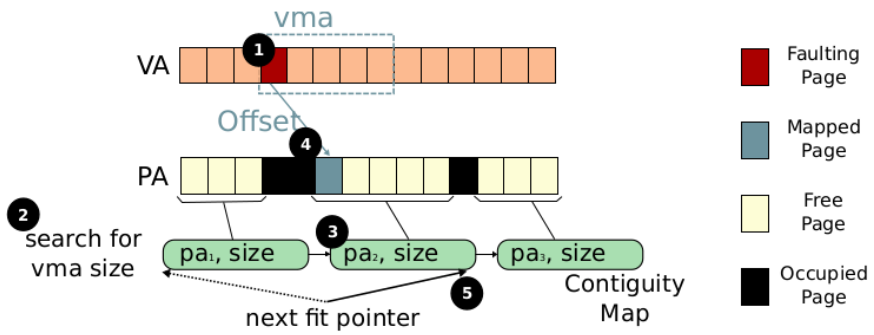
subVMA

Το subVMA είναι η δομή δεδομένων όπου χαρακτηρίζει κάθε εν δυνάμει εύρος συνεχόμενων αντιστοιχίσεων που έχει ορίσει ο μηχανισμός. Είναι ¨εν δυνάμει¨, διότι δεν έχει καμία υπάρχουσα αντιστοίχιση αρχικά και είναι πιθανό στο τέλος να μην έχουν όλες οι αντιστοιχίσεις το ίδιο Offset.

Κάθε subVMA κρατάει κάποια μεταδεδομένα, όπως την εικονική διεύθυνση που προκάλεσε το page fault που δημιούργησε το subVMA και ένα Offset. Αυτά είναι αναγκαία για τον καθορισμό, από το CaPaging, του subVMA στο οποίο ανήκει μία εικονική σελίδα και την θέση στην οποία θα πρέπει να τοποθετηθεί το φυσικό πλαίσιό της.

Κάθε VMA μπορεί να έχει περισσότερα από ένα subVMAs, αλλά σκοπός του μηχανισμού είναι η ύπαρξη όσο το δυνατόν λιγότερων subVMAs ανά VMA. Αρχικά, ο μηχανισμός δημιουργεί για κάθε VMA το πρώτο subVMA κατά το πρώτο page fault στο VMA. Η άλλη περίπτωση στην οποία δημιουργείται ένα νέο subVMA είναι όταν μία μεγάλη σελίδα αποτύχει να τοποθετηθεί στην ενδεικνυόμενη (από το CaPaging) θέση. Αυτό το νέο subVMA θα έχει διαφορετικό αντιστοιχιζόμενο εύρος φυσική μνήμης από τα υπόλοιπα subVMAs της διεργασίας.

Ο ζητούμενος χώρος στην αναζήτηση εξαρτάται από το αν υπάρχουν ήδη subVMAs στο VMA στο οποίο θα μπει το νέο subVMA. Αν δεν υπάρχουν, τότε ο ζητούμενος χώρος είναι όσο και το VMA σε μέγεθος, αλλιώς το ζητούμενο εύρος είναι από την εικονική διεύθυνση του page fault μέχρι την εικονική διεύθυνση του επόμενου (κατά αύξοντα αριθμό αρχικής εικονικής διεύθυνσης) subVMA στο VMA.



Σχήμα 2.9: *Βήματα επιλογής εύρους: 1)1ο page fault, 2) αναζήτηση στον χάρτη για ελεύθερη περιοχή, 3) επιλογή εύρους 4) ανανέωση δείκτη για επόμενη αναζήτηση. [3]*

Περιορισμοί στην ταυτόχρονη εξυπηρέτηση page faults

Κάθε VMA διατηρεί ένα spinlock για την προσπέλαση των subVMAs του. Συνεπώς, δεν είναι δυνατή η ταυτόχρονη αναζήτηση του Offset για διευθύνσεις του ίδιου VMA. Επιπλέον, διατηρεί ένα atomic flag για να επιτρέπει μόνο σε ένα νήμα την τοποθέτηση νέου subVMA. Αυτοί οι περιορισμοί τοποθετήθηκαν για να μην υπάρχει μεγάλη πίεση στην διαδικασία δημιουργίας νέων subVMA σε περίπτωση που πολλά νήματα αποτύχουν ταυτόχρονα και προσπαθήσουν συγχρόνως να δημιουργήσουν νέα subVMAs. Οπότε, σε αυτήν την περίπτωση το πρώτο νήμα έχει την δυνατότητα να δημιουργήσει νέο subVMA και τα υπόλοιπα

περιμένουν επαναλαμβάνοντας την προσπάθεια δέσμευσης ή μέχρι να έχουν την δυνατότητα τοποθέτησης νέου subVMA.

Συνοπτική περιγραφή λειτουργίας μηχανισμού

Παρακάτω περιγράφουμε σε βήματα την λειτουργία του CaPaging κατά την εξυπηρέτηση ενός page fault:

1. Αν επιτρέπεται και δεν υπάρχει subVMA στο VMA, δημιουργείται νέο subVMA.
2. Επιλέγεται το κατάλληλο subVMA για την εικονική διεύθυνση του page fault και υπολογίζεται με το Offset του subVMA η ενδεικνυόμενη θέση στην φυσική μνήμη. Στην συνέχεια ερευνάται αν η θέση είναι διαθέσιμη, και ανάλογα με το αποτέλεσμα υπάρχουν τα εξής δύο σενάρια:
   - Αν η ενδεικνυόμενη θέση είναι διαθέσιμη (δηλαδή αν ανήκει στις λίστες του buddy allocator), τότε δεσμεύεται κατευθείαν, ενώ αν ανήκει σε μεγαλύτερο διαθέσιμο block μνήμης, τότε διασπάται κατάλληλα και τα περισσευούμενα τμήματα επιστρέφουν στις λίστες του buddy allocator.
   - Αν δεν είναι διαθέσιμη, τότε αν πρόκειται για σελίδα βάσης (4KB) αποτυγχάνει. Αντιθέτως, αν πρόκειται για μεγάλη σελίδα προσπαθεί να δημιουργήσει νέο subVMA. Αν δεν βρεθεί έστω και μικρότερο εύρος στον χάρτη, τότε αποτυγχάνει. Μετά από αποτυχία του CaPaging, αναλαμβάνει ο υπάρχον μηχανισμός του λ.σ.

Επιπλέον, για να αποφεύγει το εξωτερικό κατακερματισμό λόγω δεσμεύσεων μνήμης για το PageCache, ο μηχανισμός μπορεί να διαχειρίζεται και να δεσμεύει μνήμη για αυτήν την χρήση. Αυτή η προσθήκη είναι αρκετά σημαντική σε περιπτώσεις προγραμμάτων που χρησιμοποιούν memory-mapped αρχεία, όπως το liblinear [9].

Πλεονεκτήματα - Μειονεκτήματα

Παρακάτω παρουσιάζονται τα πλεονεκτήματα και τα μειονεκτήματα του μηχανισμού (δυνατά και αδύνατα σημεία του).
Πλεονεκτήματα:

- Αρμονική σύνδεση με τον υπάρχον μηχανισμό του demand paging δίχως να τροποποιεί το πως και πότε προκαλούνται αιτήματα δέσμευσης μνήμης.
- Μικρή χρονική επιβάρυνση μόνο κατά την εξυπηρέτηση των page faults.
- Ο μηχανισμός έχει γνώση της κατάστασης της φυσικής μνήμης μέσω του contiguity map και έτσι κάνει την βέλτιστη τοποθέτηση των προς δημιουργία contiguous mappings.
- Ομαδοποιεί σε συνεχόμενες αντιστοιχίσεις και τις σελίδες του PageCache.
- Είναι δυνατή και δίχως προβλήματα η ενεργοποίηση του CaPaging σε προγράμματα που εκτελούνται παράλληλα, διότι η επιλογή ευρών στην φυσική μνήμη βασίζεται στο contiguity map.

Μειονεκτήματα:

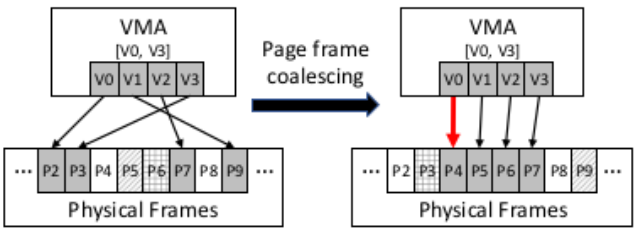
- Ο κατακερματισμός της φυσικής μνήμης δυσχεραίνει σημαντικά την επίδοση του μηχανισμού.

- Δεν μπορεί να εξυπηρετήσει παράλληλα page faults, όσον αφορά την εύρεση της κατάλληλης θέσης και την δημιουργία νέου subVMA.

2.3.3 Translation Ranger

Ο μηχανισμός Translation Ranger (TRanger ή TR) αποτελεί μία άλλη πρόταση στο ζήτημα της ανάπτυξης ενός αξιόπιστου μηχανισμού παραγωγής μεταφραστικής συνέχειας που μπορεί να αντεπεξέλθει σε όλα τα σενάρια εκτέλεσης [4]. Για να εξασφαλιστεί η αποδοτικότητα του μηχανισμού ανεξάρτητα από την κατάσταση της μνήμης (πρόβλημα κατακερματισμού), επιλέχθηκε ο μηχανισμός να τρέχει αρκετές φορές κατά την διάρκεια ζωής της διεργασίας που τον χρησιμοποιεί. Με λίγα λόγια, ο Translation Ranger[4] είναι ένας μηχανισμός που δημιουργεί όσο το δυνατόν μεγαλύτερα εύρη συνεχόμενων αντιστοιχίσεων ασύγχρονα μετακινώντας τις ήδη δεσμευμένες σελίδες της διεργασίας.

Ο μηχανισμός αφήνει ανεπηρέαστη την διαδικασία δέσμευσης φυσικής μνήμης. Η πρώτη τοποθέτηση των σελίδων στην μνήμη (εξυπηρέτηση page fault) αφήνεται να την διαχειριστεί το λ.σ. με τον δικό του μηχανισμό (buddy allocator) και ο TRanger αναλαμβάνει στην συνέχεια να δημιουργεί μεταφραστική συνέχεια με τις ήδη δεσμευμένες σελίδες, αναδιοργανώνοντας ολοκληρωτικά τις υπάρχουσες αντιστοιχίσεις μνήμης.



Σχήμα 2.10: *Παράδειγμα μετακίνησης σελίδων σε συνεχόμενες θέσεις [4]*

Anchor point

Ο μηχανισμός επιδιώκει την ύπαρξη μίας μόνο αντιστοίχισης για κάθε VMA της διεργασίας. Για να πετύχει αυτό χρησιμοποιεί ένα σημείο αναφοράς ανά VMA. Αυτό το σημείο αναφορά είναι το anchor point και πρόκειται για ένα ζευγάρι VPN-PFN (ζεύγος εικονικής και φυσικής σελίδας), βάσει του οποίου τοποθετεί τις υπόλοιπες εικονικές σελίδες που ανήκουν στο ίδιο range με αυτό. Η διαφορά του ζευγαριού του σημείου αναφοράς καθορίζει την θέση των υπόλοιπων σελίδων του ίδιου εύρους, δηλαδή πρέπει όλες οι σελίδες ενός anchor range να έχουν την ίδια διαφορά μεταξύ των VPN, PFN τους.

Τρόπος επιλογής σημείων αναφοράς(anchor points)

Τα anchor points επιλέγονται με τέτοιο τρόπο ώστε να αποφεύγεται η διασταύρωση ευρών που αντιστοιχούν σε διαφορετικά σημεία αναφοράς. Για να το πετύχει αυτό, ο TRanger καταγράφει τον χώρο φυσικών διευθύνσεων σε συμπυκνωμένες περιοχές (coalesced regions) και μη συμπυκνωμένες, και όταν δημιουργεί ένα νέο σημείο αναφοράς επιλέγει μία περιοχή που είναι coalesced region. Για να βρει μία τέτοια περιοχή διατρέχει γραμμικά όλες τις συμπυκνωμένες περιοχές και επιλέγει το πρώτο αρκετά μεγάλο κενό ανάμεσα από δύο τέτοιες περιοχές. Αν δεν βρει μία περιοχή κατάλληλη, διαγράφει το σημείο αναφοράς του μικρότερου anchor range ή απλά αγνοεί το τρέχον VMA.

Επιπλέον, αφού επιλεχτεί η περιοχή, το σημείο αναφοράς προσαρμόζεται ώστε να έχει την κατάλληλη ευθυγράμμιση το νέο range στην φυσική μνήμη. Η κατάλληλη ευθυγράμμιση είναι αυτή που επιτρέπει την χρήση μεγάλων σελίδων στο συγκεκριμένο εύρος.

Διαδικασία μετακίνησης σελίδων

Ο Translation Ranger έχει την δυνατότητα να μετακινήσει σελίδες βάσης και μεγάλες σελίδες είτε σε μία κενή θέση είτε να ανταλλάξει τις σελίδες στην φυσική μνήμη, αν η θέση είναι κατειλημμένη.

Για την μετακίνηση σελίδων σε ελεύθερο χώρο, χρησιμοποιεί τον υπάρχον μηχανισμό για migrations του πυρήνα. Όμως, το λ.σ. Linux δεν έχει την δυνατότητα να ανταλλάσσει δύο σελίδες στην φυσική μνήμη. Για αυτήν την ανάγκη, ο TRanger συνοδεύεται από έναν νέο μηχανισμό ανταλλαγής σελίδων. Αυτός ο μηχανισμός επιτρέπει μόνο την ανταλλαγή δύο ανώνυμων σελίδων ή μίας ανώνυμης με μία που συνδέεται με αρχείο (file-backed). Επιπλέον, αντί να χρησιμοποιήσει νέα σελίδα ως ενδιάμεση της αντιγραφής, μεταφέρει τα δεδομένα μεταξύ των σελίδων μέσω καταχωρητών του επεξεργαστή. Έτσι, αποφεύγεται η δέσμευση επιπλέον φυσικού χώρου και επίσης, καθίσταται δυνατή και η μετακίνηση THPs.

Ο μηχανισμός διατρέχει τον χώρο εικονικών διευθύνσεων της διεργασίας με βήμα όσο και το μέγεθος μίας σελίδας βάσης. Συγκεκριμένα, διατρέχει τα VMAs, και εσωτερικά σε κάθε VMA ελέγχει τον εικονικό χώρο, 2MB την φορά. Τέλος, αφού βρει το anchor point για το εύρος που ελέγχει, προχωρά στο έλεγχο όλων των σελίδων που το απαρτίζουν.

Ακολουθεί αναλυτικά η διαδικασία που ακολουθεί ο μηχανισμός για την διαχείριση μίας σελίδας:

1. Αρχικά, ελέγχει μέσω του πίνακα σελίδων αν για την εικονική σελίδα έχει εκχωρηθεί φυσικό πλαίσιο. Αν όχι, προχωρά στην επόμενη εικονική σελίδα.

2. Υπολογίζει την διαφορά (VPN - PFN) της τρέχουσας σελίδας. Αν είναι ίση με την διαφορά του anchor point, τότε η σελίδα είναι σωστά τοποθετημένη και προχωρά στην επόμενη σελίδα.

3. Βρίσκει την θέση που θα μπει η λανθασμένη σελίδα και ελέγχει αν είναι έγκυρη η νέα θέση της. Αν όχι, προχωρά στην επόμενη.

4. Ελέγχει αν η φυσική σελίδα προορισμού είναι ελεύθερη. Ανάλογα με την περίπτωση ο μηχανισμός εκτελεί δύο τελείως διαφορετικές διαδικασίες:

- Αν είναι ελεύθερη η θέση προορισμού, βρίσκει το block στο οποίο ανήκει στις buddy λίστες και έχουμε δύο περιπτώσεις στην συνέχεια:
  - Αν πρόκειται για σελίδα βάσης, διασπά το block και ξαναβάζει τα αχρησιμοποίητα blocks πίσω στις λίστες (αν χρειάζεται) και προχωρά στην μετακίνηση της σελίδας στην νέα θέση της χρησιμοποιώντας τον μηχανισμό για migrations του πυρήνα.
  - Αν είναι μεγάλη σελίδα, ελέγχει αν χωράει ολόκληρη στο block. Αν ναι, την μετακινεί αυτούσια πάλι με τον μηχανισμό του πυρήνα για migrations. Αν δεν χωρά, διασπά την μεγάλη σελίδα (πρόκειται για THP) σε σελίδες βάσης και πηγαίνει στο βήμα 2 της διαδικασίας, δηλαδή θα ξαναπροσπαθήσει να τις μετακινήσει (όσες από αυτές γίνεται) αλλά τώρα μία την φορά, διότι κάποιες θα μπουν σε κενές θέσεις και κάποιες άλλες σε κατειλημμένες.
- Αν είναι κατειλημμένη η θέση προορισμού, ελέγχει τι σελίδα είναι αυτή στην οποία ανήκει η θέση προορισμού (σελίδα προορισμού). Βάσει αυτού του ελέγχου, έχουμε δύο περιπτώσεις:
  - Αν και η σελίδα προς μετακίνηση και η σελίδα προορισμού είναι ίδια τάξης (και οι δύο σελίδες βάσης ή μεγάλες σελίδες), ελέγχει τον τύπο της σελίδας προορισμού. Αν είναι μετακινήσιμη και επιτρέπεται η ανταλλαγή, τότε ανταλλάσσει τα δεδομένα και τα μεταδεδομένα των φυσικών σελίδων. Αν δεν είναι, αποτυγχάνει και προχωρά στην επόμενη εικονική σελίδα η διαδικασία.
  - Αν η σελίδα προς μετακίνηση και η σελίδα προορισμού είναι διαφορετικής τάξης (η μία σελίδα βάσης και η άλλη μεγάλη), τότε αρχικά διασπά την μεγάλη σελίδα. Στην συνέχεια, αν ήταν η σελίδα προς μετακίνηση η σελίδα βάσης, προχωρά κανονικά στην ανταλλαγή των φυσικών πλαισίων για την εικονική σελίδα, αν επιτρέπεται. Αν η σελίδα προς μετακίνηση ήταν η μεγάλη σελίδα, τότε πηγαίνει στο βήμα 2 και επαναλαμβάνει την διαδικασία της ίδιας εικονικής σελίδας από την αρχή.

Επιπλέον, κάθε φορά που τελειώνει μία περιοχή εικονικών διευθύνσεων όσο και μία μεγάλη σελίδα (2MB στο x86-64), ελέγχει αν όλες οι εικονικές σελίδες της περιοχής είναι στην κατάλληλη θέση. Τότε, ενώνει τις σελίδες που το απαρτίζουν και δημιουργεί μία μεγάλη σελίδα για αυτές. Με αυτόν τον τρόπο, μειώνει τις καταχωρήσεις στον πίνακα σελίδων, βελτιώνοντας το TLB reach των κλασσικών μηχανισμών TLB.

Πλεονεκτήματα - Μειονεκτήματα

Πλεονεκτήματα:

- Επαναληπτική προσπάθεια συγκέντρωσης σελίδων σε εύρη συνεχόμενων αντιστοιχίσεων. Οπότε, αν αποτύχει μία φορά, θα έχει την ευκαιρία ο μηχανισμός να προσπαθήσει ξανά.
- Επηρεάζεται μόνο από τον κατακερματισμό που οφείλεται σε σελίδες πυρήνα. Όλες τις άλλες σελίδες (π.χ. προσωρινά κλειδωμένες σελίδες, σελίδες άλλης διεργασίας) μπορεί

να τις μετακινήσει. Επομένως, είναι αρκετά αποδοτικό στην επίτευξη μεταφραστικής συνέχειας, ανεξάρτητα πόσες διεργασίες είχαν πριν εκτελεστεί και είχαν οδηγήσει σε κατακερματισμό της μνήμης.

Μειονεκτήματα:

- Ο μηχανισμός μπορεί να μετακινήσει την ίδια σελίδα αρκετές φορές αν τροποποιηθούν τα anchor ranges του ή αν κάποιο anchor point δεν είναι πια έγκυρο (π.χ. μετά από διάσπαση ή σμίκρυνση ενός VMA).
- Οι μαζικές μετακινήσεις σελίδων δημιουργούν αξιόλογη χρονική επιβάρυνση στην εκτέλεση μίας διεργασίας, λόγω ότι το νήμα που πραγματοποιεί τις μετακινήσεις μπορεί να κρατάει κάποιο κλείδωμα αναγκαίο για κάποια άλλη λειτουργία του πυρήνα που χρειάζεται η διεργασία (π.χ. το κλείδωμα για ένα memory zone).
- Δεν είναι αποδοτική η ταυτόχρονη εκτέλεση διεργασιών που κάνουν χρήση του TRanger. Σε αντίθεση με το CaPaging, που όλα τα subVMAs που έχουν δημιουργηθεί βασίζονται στα διαθέσιμα εύρη του contiguity map, κάθε διεργασία με το TR έχει το δικό της interval tree με τα anchor ranges, οπότε η έντονη επικάλυψη ευρών διαφορετικών διεργασιών είναι δεδομένη.

Κεφάλαιο 3

Κίνητρο εργασίας

Στο κεφάλαιο αυτό θα ασχοληθούμε με την δυνατότητα ύπαρξης μίας νέας προσέγγισης στην επίτευξη μεταφραστικής συνέχειας. Για την απάντηση στο ερώτημα αυτό, βασιζόμενοι στην υπάρχουσα ερευνητική εργασία στο ζήτημα της μεταφραστικής συνέχειας, αρχικά θα αναζητήσουμε σημεία στα οποία πάσχουν οι μηχανισμοί στους οποίους έχουμε αναφερθεί στο προηγούμενο κεφάλαιο. Αφού καθορίσουμε τους παράγοντες αυτούς που καθιστούν σε κάποιες περιπτώσεις μη αποδοτική την λειτουργία των υπάρχοντων ερευνητικών προτάσεων, στην συνέχεια θα τους αναλύσουμε και θα παρουσιάσουμε το μέγεθος της επιρροής τους. Στα πλαίσια αυτής της ανάλυσης, θα στηρίξουμε τα πορίσματα μας με στοιχεία από την εκτέλεση διαφόρων πειραμάτων.

Τέλος, θα διερευνήσουμε τα περιθώρια σχεδιασμού ενός νέου μηχανισμού που θα βασίζεται στους υπάρχοντες και θα παρουσιάσουμε ένα σκελετό αυτού του συνδυασμού, λαμβάνοντας υπόψιν τους σχεδιαστικούς και λειτουργικούς περιορισμούς κάθε σχεδίασης. Στόχος αυτής της προσπάθειας είναι η εκμετάλλευση των καλύτερων στοιχείων από κάθε μηχανισμό σε μία νέα προσέγγιση που θα σχεδιαστεί με γνώμονα τους παράγοντες κακής επίδοσης δίχως όμως να θυσιάζει την καλή επίδοση σε άλλα σενάρια εκτέλεσης.

Όσον αφορά ποιους μηχανισμούς θα διερευνήσουμε, αυτοί είναι οι Contiguity-aware Paging και Translation Ranger. Αυτή η επιλογή οφείλεται στο γεγονός ότι αν και ο RMM μας προσφέρει μία πολύ χρήσιμη σχεδίαση σε επίπεδο υλικού, στα πλαίσια της διπλωματικής εργασίας ασχολούμαστε κυρίως με την ανάπτυξη ενός σχεδιασμού σε επίπεδο λογισμικού και, επιπλέον, η τεχνική του eager paging που χρησιμοποιεί ο RMM πετυχαίνει χειρότερη επίδοση από το CaPaging σχεδόν σε όλα τα πειράματα που δοκιμάστηκε [3].

3.1 Παράγοντες μειωμένης επίδοσης

Στην ενότητα αυτή, θα ασχοληθούμε με την αναζήτηση και διερεύνηση διαφόρων παραγόντων, όπου καθιστούν δυσκολότερη ή ασύμφορη την επίτευξη μεταφραστικής συνέχειας. Επιπλέον, προσπαθήσαμε να δούμε όσον τον δυνατόν πληρέστερα το ζήτημα της επίτευξης μεταφραστικής συνέχειας. Επομένως, δεν περιοριστήκαμε μόνο στο πως επηρεάζεται η μεταφραστική συνέχεια, αλλά ασχοληθήκαμε και με την επιβάρυνση που έχουν οι μηχανισμοί στους χρόνους εκτέλεσης.

Οι παράγοντες που μας ενδιαφέρουν είναι το μέγεθος του κατακερματισμού της φυσικής μνήμης, η ύπαρξη μη-μετακινήσιμων σελίδων και παράγοντες που δημιουργούν χρονική

επιβάρυνση, όπως οι μαζικές μετακινήσεις σελίδων. Για την υποστήριξη των ορισμάτων μας, θα παραθέσουμε και αντίστοιχα αποτελέσματα από κάποια πειράματα που εκτελέσαμε. Το περιβάλλον της εκτέλεσης των πειραμάτων παρουσιάζεται αναλυτικά στο Κεφάλαιο 5.

3.1.1 Κατακερματισμός μνήμης

Ένας από τους σημαντικότερους παράγοντες που οδηγούν στην ύπαρξη πάρα πολλών συνεχόμενων αντιστοιχίσεων και σε μικρά περιθώρια μεταφραστικής συνέχειας είναι ο κατακερματισμός της φυσικής μνήμης. Συγκεκριμένα, αναφερόμαστε σε εξωτερικό κατακερματισμό, ο οποίος σε μικρό βαθμό σημαίνει την ύπαρξη δεσμευμένων σελίδων που διασπούν μεγάλα εύρη συνεχόμενων διαθέσιμων σελίδων και σε μεγάλο βαθμό την ύπαρξη ελάχιστων αξιόλογα μεγάλων ευρών που δεν είναι αρκετά για να καλύψουν με λίγες συνεχόμενες αντιστοιχίσεις το αποτύπωμα στην φυσική μνήμη ενός προγράμματος που απαιτεί αρκετή φυσική μνήμη.

Περιθώρια πρόληψης φαινομένου

Πρόκειται για έναν παράγοντα του οποίου η πρόληψη είναι δύσκολη. Οι περισσότεροι μηχανισμοί στους οποίους αναφερόμαστε είναι σχεδιασμένοι να τρέχουν για μία ή λίγες διεργασίες ταυτόχρονα, οπότε δεν είναι εύκολο να ελεγχθεί ή να περιοριστεί ο κατακερματισμός που οφείλεται σε άλλες διεργασίες που έτρεχαν ή τρέχουν παράλληλα με το πρόγραμμα που κάνει χρήση ενός μηχανισμού.

Υπάρχουν τρόποι περιορισμού του κατακερματισμού, όπως ο μηχανισμός compaction του λ.σ. Linux, ο οποίος όμως εμποδίζει και την επίτευξη μεταφραστικής συνέχειας, διότι δεν έχει ως σκοπό την συνέχεια, αλλά το να συγκεντρώσει όλες τις σελίδες σε συνεχόμενες θέσεις στην μία πλευρά της φυσικής μνήμης, ώστε να αφήσει έναν μεγάλο χώρο διαθέσιμο στην άλλη πλευρά. Θα μπορούσε να διερευνηθεί και μία τροποποίηση αυτού του μηχανισμού ώστε να αντιλαμβάνεται την συνέχεια που έχει επιτευχθεί και να μην την αλλοιώνει, όμως αυτό παρεκκλίνει από το αντικείμενο της συγκεκριμένης διεργασίας και αποτελεί μία προσέγγιση αποκλειστικά χρήσιμη για το λ.σ. Linux.

Ένα ακόμα εμπόδιο στον περιορισμό του κατακερματισμού της μνήμης είναι η ύπαρξη μόνιμα μη-μετακινήσιμων σελίδων. Για τις ανάγκες των λειτουργιών του πυρήνα δεσμεύεται ένας μη αμελητέος αριθμός σελίδων, ο οποίος δεν μπορεί να μετακινηθεί και αν βρεθεί σε θέση μέσα σε ένα μεγάλο διαθέσιμο εύρος μνήμης, μέχρι να αποδεσμευτεί η σελίδα θα δημιουργεί μόνιμο κατακερματισμό, ο οποίος δεν μπορεί να αντιμετωπιστεί αφού συμβεί. Περισσότερες πληροφορίες για αυτές τις σελίδες θα αναφέρουμε στην συνέχεια, αφού έχουμε αφιερώσει μία ολόκληρη υποενότητα στο ζήτημα των μη-μετακινήσιμων σελίδων.

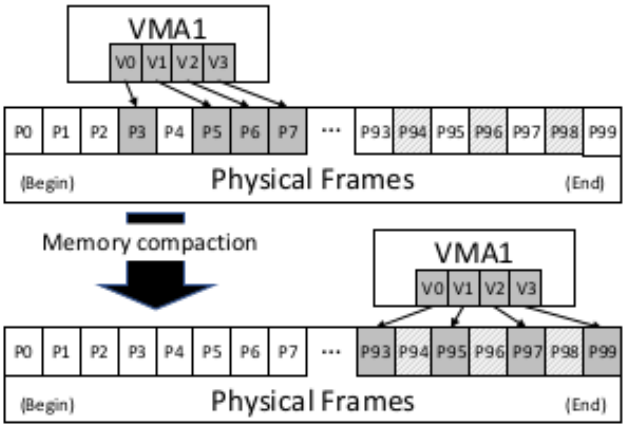
Κατακερματισμός και Contiguity-aware Paging

Ο CaPaging για την επιλογή των ranges βασίζεται στα στοιχεία του contiguity map. Όμως, ο contiguity map επί της ουσίας αποτυπώνει τον κατακερματισμό της μνήμης. Συνεπώς, η επίδοση του CaPaging εξαρτάται σημαντικά από τον κατακερματισμό της μνήμης, διότι ένας μεγάλος κατακερματισμός του contiguity map σημαίνει πολλά μικρά διαθέσιμα ranges, το οποίο οδηγεί στην δημιουργία πολλών subVMAs.

Ένα σενάριο που εμφανίζεται, λόγω κατακερματισμού, είναι ο μηχανισμός να ζητά συνεχώς μεγάλα ranges για τα νέα subVMAs που θέλει να δημιουργήσει, αλλά να μην υπάρχουν αρκετά μεγάλα διαθέσιμα εύρη στον χάρτη. Αυτό έχει ως αποτέλεσμα ανά αρκετές σελίδες να ζητάει ξανά χώρο για νέο subVMA, διότι το προηγούμενο δεν είναι ικανό να του τον παρέχει.

Επίσης, ένα άλλο πρόβλημα που δημιουργεί ο κατακερματισμός (ακόμα και σε μικρό βαθμό) είναι ένα μεγάλο range να διασπαστεί σε μερικά τμήματα από πολύ λίγες σελίδες. Τότε, χάνεται ένα πολύτιμο μεγάλο εύρος για ελάχιστες σελίδες, το οποίο δεν μπορεί κάπως να ανακτηθεί εύκολα ακόμα και αν είναι μετακινήσιμες σελίδες, διότι δεν υπάρχει μηχανισμός compaction που να μην αλλοιώνει και την μεταφραστική συνέχεια (Σχήμα 3.1).

Με τα αποτελέσματα που παραθέτουμε στα διαγράμματα 3.2α΄ και 3.2β΄, επιβεβαιώνουμε ότι ο κατακερματισμός της μνήμης δυσχεραίνει το έργο του CaPaging, και όσο αυξάνεται ο βαθμός τόσο χειρότερη επίδοση έχουμε.



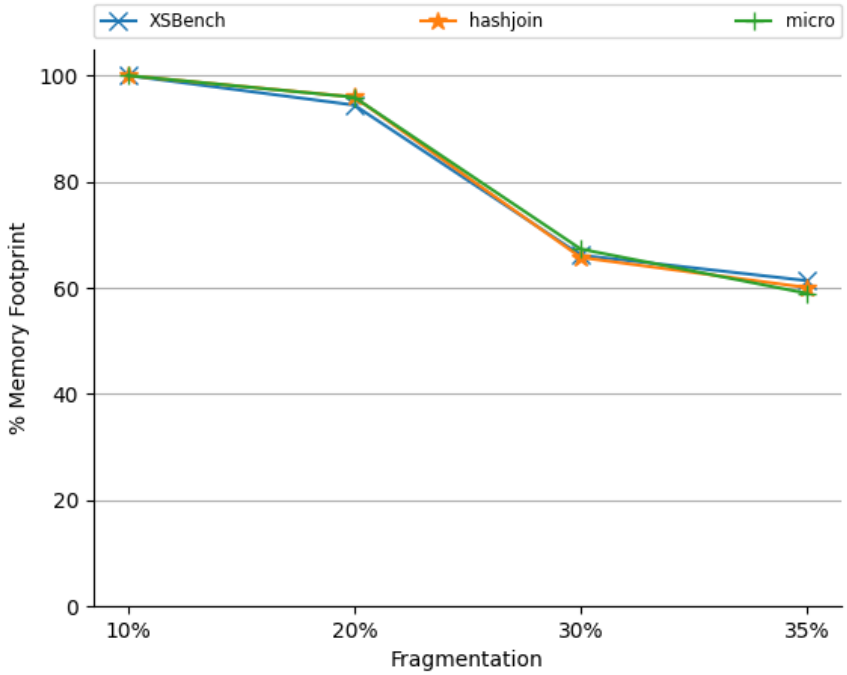
Σχήμα 3.1: *Παράδειγμα αλλοίωση συνέχειας λόγω compaction. [4]*

Κατακερματισμός και Translation Ranger

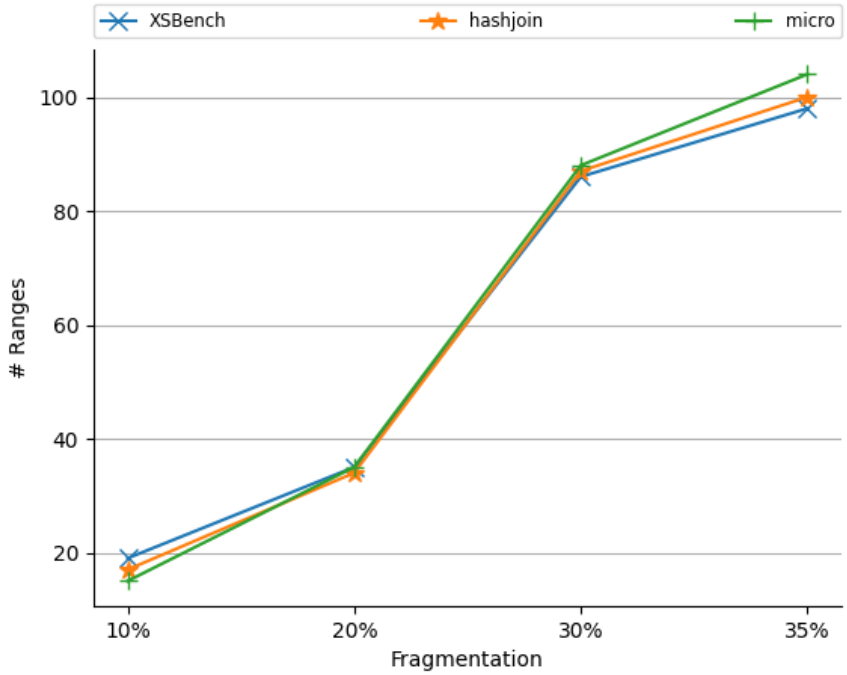
Ο Translation Ranger μετακινεί σελίδες αφού έχουν δεσμευτεί σε μία τυχαία θέση και δεν λαμβάνει υπόψιν την κατάσταση της φυσικής μνήμης για τα ranges που δημιουργεί. Λόγω αυτής της φύσης του μηχανισμού, δεν επηρεάζεται σημαντικά από τυχόν υφιστάμενο κατακερματισμό της μνήμης.

Επηρεάζεται, όμως, από την σχέση κατακερματισμού και μη-μετακινήσιμων σελίδων. Λόγω κάποιου πρότερου κατακερματισμού ή λόγω της τυχαιότητας στην επιλογή θέσης για τις σελίδες (buddy allocator), μπορεί οι μη-μετακινήσιμες σελίδες να έχουν τοποθετηθεί σε πολλά διαφορετικά μη συνεχόμενα σημεία στην φυσική μνήμη. Αυτό δημιουργεί έναν κατακερματισμό που δεν μπορεί να τον προσπεράσει ο Translation Ranger διότι ο μηχανισμός δεν ασχολείται με το τι υπάρχει στην μνήμη στα σημεία που επιλέγει να συγκεντρώσει τις σελίδες σε συνεχόμενες θέσεις.

Όμως, το παραπάνω σενάριο δεν είναι αρκετό για να οδηγήσει σε χειρότερα αποτελέσματα. Μάλιστα, στα πειράματα που εκτελέσαμε (διαγράμματα 3.3), η επίδοση του αλγορίθμου παρέμεινε σε παρόμοια επίπεδα ανεξάρτητα από τον βαθμό κατακερματισμού της φυσικής μνήμης.

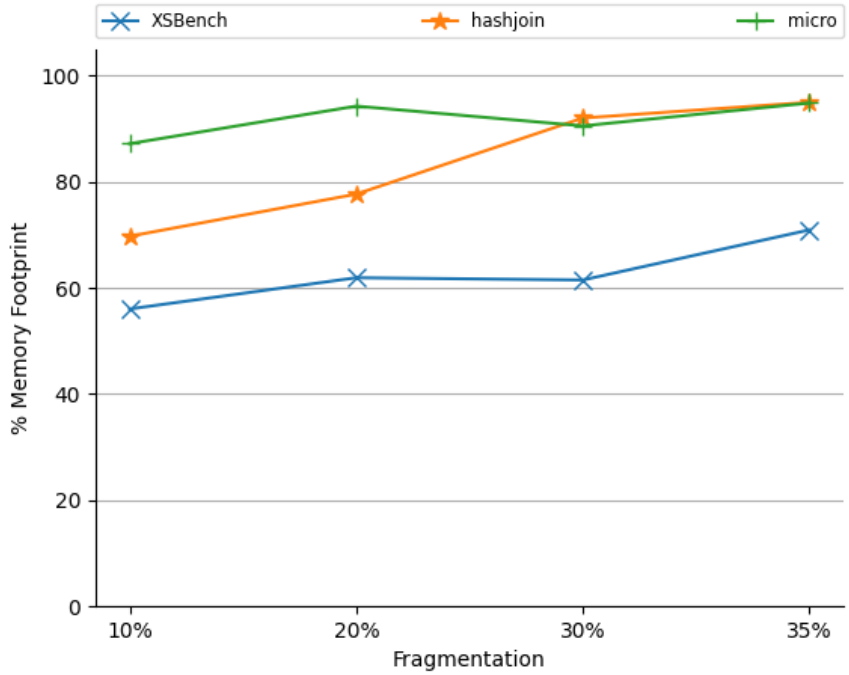


(α') *Κάλυψη με τα 32 μεγαλύτερα mappings*

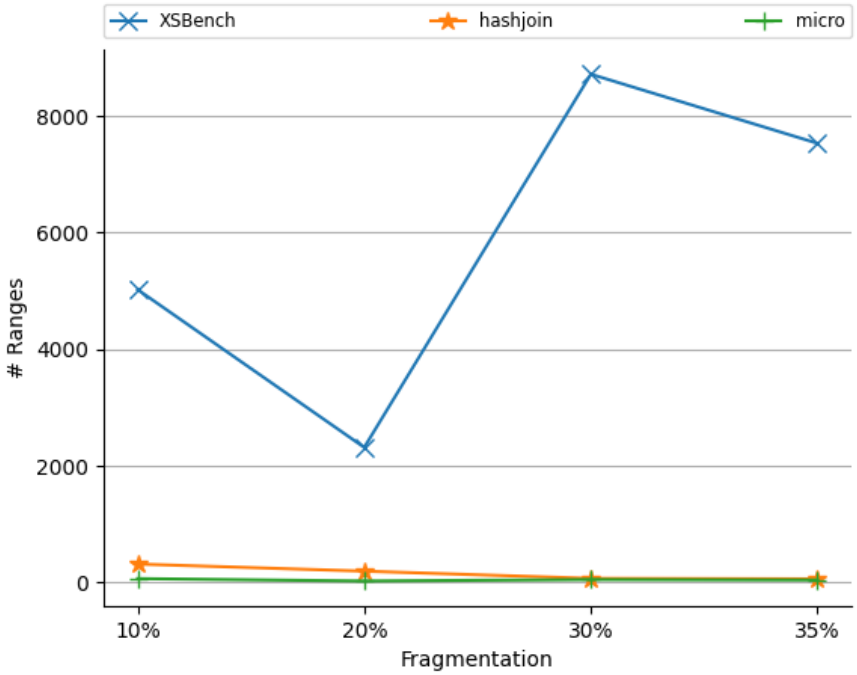


(β') *Πλήθος mappings για την κάλυψη του 99% της μνήμης*

Σχήμα 3.2: *Επίδοση του* ***CaPaging*** *για διαφορετικά ποσοστά κατακερματισμού*



(α') *Κάλυψη με τα 32 μεγαλύτερα mappings*



(β') *Πλήθος mappings για την κάλυψη του 99% της μνήμης*

Σχήμα 3.3: *Επίδοση του* ***Translation Ranger*** *για διαφορετικά ποσοστά κατακερματισμού*

3.1.2 Μαζικές μετακινήσεις σελίδων

Αυτός ο παράγοντας αφορά αποκλειστικά τον μηχανισμό του Translation Ranger. Για να πετύχει την μεταφραστική συνέχεια, ο TRanger χρειάζεται να μετακινήσει όλες τις υπάρχουσες σελίδες σε θέσεις που καθορίζονται κατά την εκτέλεσή του νήματός του. Αυτό οφείλεται στο γεγονός ότι ο TRanger δεν έχει λόγο στο που θα πρωτοτοποθετηθούν οι σελίδες, αφού η δέσμευσή τους γίνεται από τον υπάρχον μηχανισμό δέσμευση μνήμης του λ.σ., που τις τοποθετεί σε τυχαίες θέσεις στην φυσική μνήμη.

Όλες αυτές οι μετακινήσεις επιβαρύνουν σημαντικά την εκτέλεση του προγράμματος, διότι για κάθε μετακίνηση ο μηχανισμός του TRanger χρειάζεται να κάνει κάποιες ενέργειες που επιβαρύνουν την φυσική μνήμη και καθυστερούν άλλες λειτουργίες του πυρήνα (π.χ. λόγω δέσμευσης κλειδωμάτων). Αυτό επιβεβαιώνεται και από τον πίνακα 3.1 που παρουσιάζει τους χρόνους εκτέλεσης των δύο μηχανισμών σε διάφορα πειράματα.

Benchmark	Μηχανισμός	User (ms)	System (ms)
Micro	CaPaging	124032	37441
	TRanger	145445	55649
XSBench	CaPaging	286248	40601
	TRanger	365055	49456
Hashjoin	CaPaging	1876660	49516
	TRanger	1910198	71388

Πίνακας 3.1: *Μέσοι χρόνοι πειραμάτων*

Αυτή η επιβάρυνση δεν είναι αμελητέα και ένας από τους λόγους που υφίσταται είναι το γεγονός ότι η μετακίνηση κάθε σελίδας δεν γίνεται απαραίτητα μόνο μία φορά. Κάθε φορά που ένα VMA μεταβάλλεται (π.χ. επεκτείνεται, διασπάται ή ενώνεται με κάποιο άλλο) είναι πολύ πιθανό ο TRanger να επαναορίσει τα anchor ranges του VMA. Ως αποτέλεσμα, σελίδες που είχαν μετακινηθεί, θα χρειαστεί να αλλάξουν ξανά θέση. Λαμβάνοντας μετρήσεις για το πλήθος των μετακινήσεων που έκανε ο μηχανισμός σε κάποια πειράματα, παρατηρήσαμε (Πίνακας 3.2) ότι όχι μόνο το συνολικό πλήθος των δεδομένων που μετακινήθηκαν ήταν περισσότερο, αλλά 2 έως 4 φορές μεγαλύτερο από το σύνολο της μνήμης που χρησιμοποιήθηκε σε κάθε πρόγραμμα.

Benchmark	Σύνολο δεσμευμένης μνήμης	Επιτυχημένες μετακινήσεις	Αποτυχημένες μετακινήσεις
Micro	120GB	310.8GB	13.2GB
XSBench	116.5GB	176GB	28.7GB
Hashjoin	116.6GB	413.3GB	21.6GB

Πίνακας 3.2: *Πλήθος μετακινήσεων σελίδων λόγω Translation Ranger (σε GB)*

3.1.3 Μη-μετακινήσιμες σελίδες

Ένας ακόμα παράγοντας που προαναφέρθηκε είναι η ύπαρξη μη-μετακινήσιμων σελίδων. Τέτοιες σελίδες είναι οι σελίδες του πυρήνα ή σελίδες που ο πυρήνας έχει κλειδώσει στην συγκεκριμένη θέση προσωρινά (π.χ. κάποια Ε/Ε ενέργεια). Ειδικά, οι πρώτες είναι

αρκετά δύσκολο να μετακινηθούν και συχνά παραμένουν για αρκετή ώρα δεσμευμένες, επειδή περιέχουν πίνακες σελίδων, slab αντικείμενα και άλλα σημαντικά για τον πυρήνα δεδομένα. Μάλιστα, αν αυτές είναι διάσπαρτες στην φυσική μνήμη, ο κατακερματισμός που δημιουργούν, είναι αδύνατον να περιοριστεί και δημιουργεί πρόβλημα και στους δύο μηχανισμούς.

Συγκεκριμένα, ο CaPaging δεν επηρεάζεται άμεσα από αυτές, αλλά έμμεσα αν αυτές οι σελίδες προκαλέσουν εξωτερικό κατακερματισμό. Ο μηχανισμός του CaPaging τις βλέπει ως μη διαθέσιμες σελίδες και αν πρόκειται για δέσμευση μεγάλης σελίδας, απλά προχωράει στην δημιουργία νέου subVMA.

Αντιθέτως, όσον αφορά το Translation Ranger, η αποτελεσματικότητά του εξαρτάται από τον τύπο της μη-μετακινήσιμης σελίδας. Οι προσωρινά μη-μετακινήσιμες σελίδες αντιμετωπίζονται ικανοποιητικά, διότι αν αποτύχει μία φορά να τις μετακινήσει, θα προσπαθήσει ξανά σε επόμενο χρόνο και θα πετύχει. Δεν ισχύει όμως το ίδιο για τις σελίδας του πυρήνα, στην θέση των οποίων δεν μπορεί να μετακινήσει μία σελίδα.

3.2 Μία νέα πρόταση προς μία πιο ολοκληρωμένη επίτευξη συνέχειας

3.2.1 Διερεύνηση σεναρίου συνδυασμού των δύο μηχανισμών

Αξίζει λοιπόν κάτι καινούργιο;

Στους μηχανισμούς που αναλύσαμε, αν και έχουν γενικά ικανοποιητική συμπεριφορά, υπάρχουν κάποια ζητήματα που δυσχεραίνουν την απόδοση τους. Στην περίπτωση του CaPaging είχαμε τον κατακερματισμό μνήμης που τον εμπόδιζε να δημιουργήσει υψηλή μεταφραστική συνέχεια. Αντιθέτως, στην περίπτωση του TRanger το κυρίως πρόβλημα παρουσιαζόταν σε κάθε σενάριο εκτέλεσης και αφορούσε την χρονική επιβάρυνση που δημιουργούν οι ασύγχρονες μαζικές μετακινήσεις σελίδων. Είχε και ο TRanger μικρό θέμα με τον κατακερματισμό της μνήμης, αλλά μόνο όπου οφειλόταν σε διάσπαρτες μόνιμες μη-μετακινήσιμες σελίδες. Επομένως, από την στιγμή που αυτοί οι παράγοντες δεν είναι αμελητέοι, θεωρήσαμε χρήσιμη την αναζήτηση ενός σχεδιασμού που αντιμετωπίζει κατάλληλα αυτά τα ζητήματα, δίχως να δημιουργεί προβλήματα σε άλλες περιπτώσεις.

CaP με TR: ένας αισιόδοξος συνδυασμός

Πριν όμως προχωρήσουμε στην ανάλυση του νέου σχεδιασμού, θα εξηγήσουμε γιατί είναι καλή και βολική η ιδέα του συνδυασμού τους. Αναλυτικά, οι λόγοι που μας προθυμοποίησαν για την υλοποίηση του συνδυασμού των μηχανισμών:

- Παρατηρήσαμε ότι όπου ο ένας μηχανισμός πάσχει, ο άλλος δουλεύει αποδοτικά. Συγκεκριμένα, ο CaPaging επιτυγχάνει δίχως σημαντική χρονική επιβάρυνση στην επίτευξη συνέχειας λόγω ότι δεν επιβαρύνει σημαντικά τον μηχανισμό δέσμευσης μνήμης, ενώ ο TRanger λόγω των μαζικών μετακινήσεων δημιουργεί μία χρονική επιβάρυνση σε κάθε σενάριο εκτέλεσης που ερευνήσαμε. Αντιθέτως, ο TRanger έχει πολύ

καλή απόδοση σε εκτελέσεις με αρκετά κατακερματισμένη μνήμη, ενώ ο CaPaging όσο πιο κατακερματισμένη είναι η μνήμη, τόσο πιο κακή επίδοση έχει.

- Οι μηχανισμοί ενεργούν σε διαφορετικές φάσεις κατά την ζωή μία σελίδας. Ο CaPaging επηρεάζει την φάση δέσμευσης μία σελίδας, ενώ ο TRanger ασχολείται με την σελίδα αφού ήδη έχει δεσμευτεί από το λ.σ.

- Τροποποιούν και επηρεάζουν διαφορετικά την λειτουργία του πυρήνα. Ο CaPaging αποτελεί τμήμα του μηχανισμού δέσμευσης μνήμης του πυρήνα και λειτουργεί κατά την εξυπηρέτηση των page faults. Αντιθέτως, ο TRanger αφήνει απείραχτο τον μηχανισμό δέσμευσης μνήμης και απλά έχει ένα νήμα πυρήνα που μετακινεί σελίδες που ήδη έχουν δεσμευτεί. Επίσης, και οι δύο μηχανισμοί έχουν τα δικά τους ανεξάρτητα μεταδεδομένα. Επομένως, δεν υπάρχει κάποια διασταύρωση στα σημεία που κάθε μηχανισμός έχει τροποποιήσει στο λ.σ. .

Σχεδιαστικές-λειτουργικές απαιτήσεις από τον νέο μηχανισμό

Σκοπός του νέου μηχανισμού θα πρέπει να είναι η επίτευξη μεταφραστικής συνέχειας, καθ όλη τη διάρκεια ζωής των σελίδων της διεργασίας και ανεξάρτητα από την κατάσταση της φυσικής μνήμης. Συγκεκριμένα από τον νέο μηχανισμό ζητούνται τα εξής χαρακτηριστικά:

- Ευελιξία και αρμονική σύνδεση με τον μηχανισμό δέσμευσης σελίδων του λ.σ.

- Χαμηλή επιβάρυνση στον χρόνο εκτέλεσης των διεργασιών που τον χρησιμοποιούν.

- Προσπάθεια για δημιουργία μεγάλων contiguous mappings σε όσες περισσότερες φάσεις στην ζωή μίας σελίδας της διεργασίας. Με άλλα λόγια, να μην περιορίζεται η σωστή τοποθέτηση μόνο κατά την δέσμευση της σελίδας ή μόνο μετακινώντας την μετά.

- Καλύτερη αρχική τοποθέτηση σελίδων σε κατακερματισμένη φυσική μνήμη.

- Από την στιγμή που θα μπορεί να μετακινεί σελίδες, θα πρέπει να περιοριστούν όσο το δυνατόν οι μετακινήσεις σελίδων.

3.2.2 Η πρότασή μας: ο μηχανισμός MACAP

Η πρόταση αυτής της διπλωματικής (και το κεντρικό έργο της) για την ικανοποίηση των παραπάνω απαιτήσεων είναι ο μηχανισμός Migration-assisted Contiguity-aware Paging ή αλλιώς MACAP. Πρόκειται για τον συνδυασμό των CaPaging και TRanger που αναφέραμε. Βασίζεται στην παρατήρηση ότι οι υπάρχοντες μηχανισμοί δρουν σε διαφορετικές φάσεις, έχοντας συγχρόνως και διαφορετική προσέγγιση στον τρόπο επίτευξης συνέχειας.

Όπως και οι μηχανισμοί στους οποίους βασίζεται, ο MACAP δημιουργεί εύρη συνεχόμενων αντιστοιχίσεων. Αυτά τα εύρη μετά μπορεί να τα εκμεταλλευτεί ένας μηχανισμός υλικού που έχει την δυνατότητα μετάφρασης διευθύνσεων κάνοντας χρήση ευρών αντί μόνο σελίδων. Τέτοιος μηχανισμός, που είναι κατάλληλος να συνεργαστεί με το MACAP είναι ο RMM[2] (δίχως το eager paging) ή οποιοσδήποτε άλλος μηχανισμός εκμεταλλεύεται την ιδέα των contiguous mappings.

Στον νέο μηχανισμό, οι δύο μηχανισμοί κάνουν ότι έκαναν και μεμονωμένοι, αλλά πια έχουν κοινό στόχο για το σημείο τοποθέτησης κάθε σελίδας. Ο CaPaging ασχολείται με την τοποθέτηση της σελίδας στην ενδεικνυόμενη θέση κατά την δέσμευση της σελίδας, και ο TRanger προσπαθεί να μετακινήσει τις σελίδες που δεν είναι στην κατάλληλη θέση. Το ποια είναι η κατάλληλη θέση καθορίζεται κατά την δέσμευση της σελίδας,δηλαδή κατά την λειτουργία του CaPaging, διότι ασχολείται με τις σελίδες όταν πρωτοδεσμεύονται. Όσες σελίδες αποτυγχάνουν στο CaPaging, τις αναλαμβάνει ο TRanger. Επομένως, ο TRanger ασχολείται αποκλειστικά με τις σελίδες που δεν τοποθετήθηκαν επιτυχώς από το CaPaging και σκοπός του είναι να επαναφέρει την συνέχεια μετακινώντας σελίδες στις θέσεις που τους είχαν ενδεικνυθεί κατά τη δέσμευσή τους.

Με αυτό τον τρόπο, δεν πάει χαμένη καμία ευκαιρία σε οποιαδήποτε φάση της ¨ζωής¨ μίας δεσμευμένης σελίδας. Επιδιώκεται η κατάλληλη τοποθέτησή της από την πρώτη στιγμή που δεσμεύεται και αν αποτύχει αρχικά, μετά αποτελεί ευθύνη του TRanger η επανατοποθέτησή της, το οποίο το προσπαθεί μέχρι και το τέλος της διεργασίας στην οποία ο μηχανισμός είναι ενεργοποιημένος.

Τον αρχικό αυτό συνδυασμό πλαισιώνουν προσθήκες χρήσιμες και αναγκαίες για την εκμετάλλευση των νέων δυνατών σημείων του συνδυασμού τους. Ο κλασσικός CaPaging όταν αποτυγχάνει να τοποθετήσει μία μεγάλη σελίδα, καταφεύγει στην δημιουργία νέου subVMA. Επομένως, αφήνει μόνο σελίδες βάσης (4KB) να αποτύχουν, όταν υπάρχουν διαθέσιμα εύρη στον contiguity map. Αυτή η συμπεριφορά δεν αφήνει και αρκετές σελίδες για να μετακινήσει ο TRanger, και ούτε περιορίζει τον αριθμό των subVMAs που δημιουργούνται. Για τον περιορισμό του πλήθος των subVMA, ο CaPaging μπορεί να αφήσει έναν αριθμό μεγάλων σελίδων να αποτύχουν αν είναι δεσμευμένη ήδη η θέση, ώστε να αποφύγει να δημιουργήσει νέο subVMA και να αναλάβει την τοποθέτησής τους ο TRanger που μπορεί να μετακινεί και να εναλλάσσει την θέση δύο σελίδων.

Περισσότερες πληροφορίες για την υλοποίηση και συμπεριφορά του νέου μηχανισμού παραθέτουμε στο επόμενο κεφάλαιο, ενώ η αξιολόγησή του γίνεται στο Κεφάλαιο 5.

Κεφάλαιο 4

Υλοποίηση

Στο κεφάλαιο αυτό περιγράφεται η υλοποίηση του μηχανισμού **Migration-assisted Contiguity-aware Paging** (MACAP), που αποτελεί τον συνδυασμό των Contiguity-aware Paging και Translation Ranger. Θα παρουσιαστούν οι διάφορες σχεδιαστικές επιλογές που έγιναν, καθώς και κάποιες τελικές προσθήκες για την μεγαλύτερη αποτελεσματικότητα του τελικού μηχανισμού.

4.1 Περιγραφή βασικού μηχανισμού

Ο MACAP αποτελεί αποτέλεσμα της προσπάθειας για την δημιουργία ενός μηχανισμού λογισμικού, στο περιβάλλον του πυρήνα του Linux, που έχει ως στόχο την συνεχή και πολύπλευρη βελτίωση της μεταφραστικής συνέχειας μίας διεργασίας.

Ο MACAP έχει ως πυρήνα της υλοποίησής του τον CaPaging, με τον TRanger να δρα επικουρικά με τα migrations. Αναλυτικότερα, η επιλογή της θέσης των νέων σελίδων γίνεται μέσω του μηχανισμού του CaPaging κατά την εξυπηρέτηση των page faults ενώ ο μηχανισμός του TRanger χρησιμοποιείται αποκλειστικά για την μετακίνηση όσων σελίδων απέτυχαν να μπουν ,κατά το allocation τους, στην ενδεδειγμένη θέση. Οι θέσεις στις οποίες τοποθετούνται οι σελίδες στην φυσική μνήμη καθορίζονται από τα μεταδεδομένα (subVMAs) που δημιουργούνται και τροποποιούνται αποκλειστικά στα πλαίσια του (τροποποιημένου) μηχανισμού του CaPaging, ενώ ο TRanger δεν έχει λόγο στον καθορισμό των ranges για τις φυσικές σελίδες της διεργασίας.

Όπως είναι εμφανές από την παραπάνω περιγραφή, οι μηχανισμοί δεν χρησιμοποιούνται αυτούσιοι, αλλά έχουν γίνει αρκετές τροποποιήσεις και προσθήκες ώστε να ταιριάξουν και να γίνει σωστή εκμετάλλευση των νέων δυνατοτήτων που προκύπτουν από την συνύπαρξη τους. Επίσης, οι τροποποιήσεις-προσθήκες που περιγράφονται παρακάτω αφορούν μεγάλες σελίδες που ανήκουν σε VMAs, ενώ για λοιπές περιπτώσεις (PageCache) δεν αλλάζει η διαδικασία εκχώρησης μνήμης.

4.2 Μεταδεδομένα

Η ενοποίησή των CaP και TR σε έναν κοινό μηχανισμό απαιτούσε κοινά μεταδεδομένα, ώστε να δρουν με τον ίδιο τρόπο κατά την επιλογή της σωστής φυσικής θέσης για μια σελίδα. Οι δύο μηχανισμοί κρατούν παρόμοιες πληροφορίες για τα ranges που φτιάχνουν, επομένως

η απλούστερη και λιγότερο επιβαρυντική λύση ήταν να διατηρηθούν οι δομές δεδομένων του ενός μηχανισμού και ο άλλος να προσαρμοστεί στην προσπέλαση και αξιοποίηση αυτών.

Για τον συνδυασμό τους επιλέχθηκαν οι δομές δεδομένων του CaPaging. Η επιλογή αυτή έγινε λόγω του ότι για κάθε σελίδα που υπάρχει στην μνήμη, πρώτα τρέχει ο CaPaging και σε επόμενο χρόνο ασχολείται μαζί της ο τροποποιημένος TRanger. Με άλλα λόγια, αφού ο ρόλος του TRanger είναι να διορθώσει αποτυχίες του CaPaging, είναι εύλογο να κρατήσουμε τα μεταδεδομένα του CaPaging, στα πλαίσια του οποίου δημιουργούνται νέα ranges (subVMAs) και υποδεικνύονται οι θέσεις στην φυσική μνήμη για τις νέες σελίδες. Για να λειτουργεί ο μηχανισμός του TRanger με τα μεταδεδομένα του CaPaging, αφαιρέθηκε η δυνατότητα από το TRanger να κρατάει τα δικά του μεταδεδομένα και τροποποιήθηκε κατάλληλα ώστε να μπορεί να διατρέχει τα subVMAs για να βρίσκει το αντίστοιχο για μία ζητούμενη εικονική διεύθυνση.

4.3 Συντηρητική προσέγγιση στην δημιουργία νέων subVMAs

Η κλασική συμπεριφορά του CaPaging ,όταν είναι δυνατή η δημιουργία νέου subVMA, είναι να ψάχνει στο contiguity map για διαθέσιμο χώρο και να δημιουργεί νέο subVMA σε κάθε αποτυχία τοποθέτησης μίας μεγάλης σελίδας στην ενδεδειγμένη θέση. Σε αντίθεση με την παλιά πολιτική δημιουργίας νέων subVMAs, ο νέος μηχανισμός είναι πιο συντηρητικός στην δημιουργία νέων subVMAs. Αυτό το πραγματοποιεί με το να αφήνει, κατά την εκχώρηση μνήμης, κάποιες μεγάλες σελίδες να αποτύχουν (εναλλαγή στον κλασικό μηχανισμό δέσμευσης μνήμης) δίχως να δημιουργεί νέο subVMA.

4.3.1 Περιγραφή Λειτουργίας

Κάθε subVMA διατηρεί το δικό του μετρητή για τις αποτυχίες τοποθέτησης μεγάλων σελίδων στο εύρος που του αναλογεί. Όταν σε ένα subVMA το πλήθος αυτών φτάσει ένα συγκεκριμένο όριο (ίδιο system-wide) , ο μηχανισμός δημιουργεί ένα νέο subVMA που ξεκινά από την τρέχουσα εξυπηρετούμενη εικονική διεύθυνση και μηδενίζει τον μετρητή του παλιού subVMA. Αντιθέτως, όταν υπάρξει μέσα σε ένα subVMA επιτυχημένη τοποθέτηση μιας μεγάλης σελίδας, ο μετρητής του subVMA μηδενίζεται.

4.3.2 Παράγοντες αποτελεσματικότητας

Ο βαθμός της αποτροπής δημιουργίας νέων subVMAs καθορίζεται από το ύψος του ορίου που αναφέρθηκε παραπάνω. Για πολύ μικρές τιμές, ο CaPaging δουλεύει όπως στην αρχική του έκδοση (πολλά subVMAs), ενώ για πολύ μεγάλες αφήνεται πολύ περισσότερη δουλειά στον μηχανισμό του TRanger. Επίσης, το όριο χρειάζεται να είναι αρκετά μεγάλο για να καλύπτει πλήρως τα περισσότερα συνεχή τμήματα στην φυσική μνήμη, όπου δεν υπάρχει εσωτερικά κανένα ελεύθερο συνεχές block, μεγέθους $\geqslant 2MB$.

Πέρα από την τιμή, η δυνατότητα αποτροπής δημιουργίας ενός νέου range εξαρτάται και από τον τύπο των σελίδων που δεσμεύουν τις ενδεικνυόμενες θέσεις από το CaPaging. Ο MACAP επωφελείται από αυτήν την λειτουργία όταν πρόκειται για βραχύβιες σελίδες του πυρήνα, για απασχολημένες σελίδες και για σελίδες άλλων διεργασιών.

4.3.3 Σκοπός

Ο σκοπός αυτής της πιο συντηρητικής προσέγγισης είναι ο περιορισμός του αριθμού των subVMAs, αφήνοντας κάποιες σελίδες να τις βάλει έπειτα στην σωστή θέση ο TRanger. Έτσι, μειώνοντας τα subVMAs και παράλληλα προσπαθώντας να διορθωθεί μέσω του TRanger το κενό στην μεταφραστική συνέχεια που δημιουργήθηκε προσωρινά, σκοπεύουμε στην καλύτερη κάλυψη με λιγότερα ranges του αποτυπώματος της διεργασίας στην φυσική μνήμη.

4.4 Μαρκάρισμα λανθασμένα τοποθετημένων σελίδων

Κάθε σελίδα που αποτυγχάνει να τοποθετηθεί στην ενδεικνυόμενη θέση στην μνήμη, σημαδεύεται με ένα νέο page flag. Το νέο page flag ονομάζεται *Misplaced* και το αποκτά μία φυσική σελίδα που ο μηχανισμός του CaPaging δεν κατάφερε να τοποθετήσει κατάλληλα. Το flag αυτό αφαιρείται όταν η σελίδα μετακινηθεί στην θέση που της υποδεικνύεται από το subVMA στο οποίο ανήκει η εικονική της διεύθυνση.

Η προσθήκη ενός page flag, αν και δεν είναι η βέλτιστη πρακτική, δεν επιβαρύνει αρκετά το διαθέσιμο χώρο για flags, διότι στα 64bit συστήματα ο διαθέσιμος χώρος για flags είναι αρκετός. Επίσης, η χρήση του στον μηχανισμό προσφέρει αρκετά οφέλη και είναι ένας καλός και γρήγορος τρόπος να γνωρίζει ο TRanger για ποιες σελίδες έχει αποτύχει ο CaPaging, διότι σε σχέση με την στιγμή της δέσμευσης, όταν θα τρέχει ο TRanger, ο \χάρτης" των subVMA μπορεί να έχει αλλάξει (προσθήκη νέων subVMAs).

4.5 Προληπτική δημιουργία νέου subVMA

Ένα υψηλό όριο στον αριθμό των σελίδων που αφήνονται να αποτύχουν πριν την δημιουργία νέου subVMA οδηγεί σε έναν υψηλό αριθμό λανθασμένα τοποθετημένων σελίδων. Οι σελίδες αυτές, λόγω ότι τοποθετούνται από τον buddy allocator, μπορεί να οδηγήσουν σε περαιτέρω κατακερματισμό του contiguity map και της μνήμης. Επιπλέον, για τις σελίδες αυτές θα καταναλωθούν σημαντικοί υπολογιστικοί πόροι κατά τις απόπειρες μετακίνησής τους από το TRanger. Για την αποφυγή αυτών των συνεπειών, εισάγαμε στον CaPaging έναν απλοϊκό μηχανισμό που προβαίνει στην προληπτική δημιουργία νέου subVMA όπου προβλέπει ότι θα ακολουθήσουν συνεχόμενες και μαζικές λανθασμένες τοποθετήσεις μεγάλων σελίδων.

4.5.1 Περιγραφή λειτουργίας

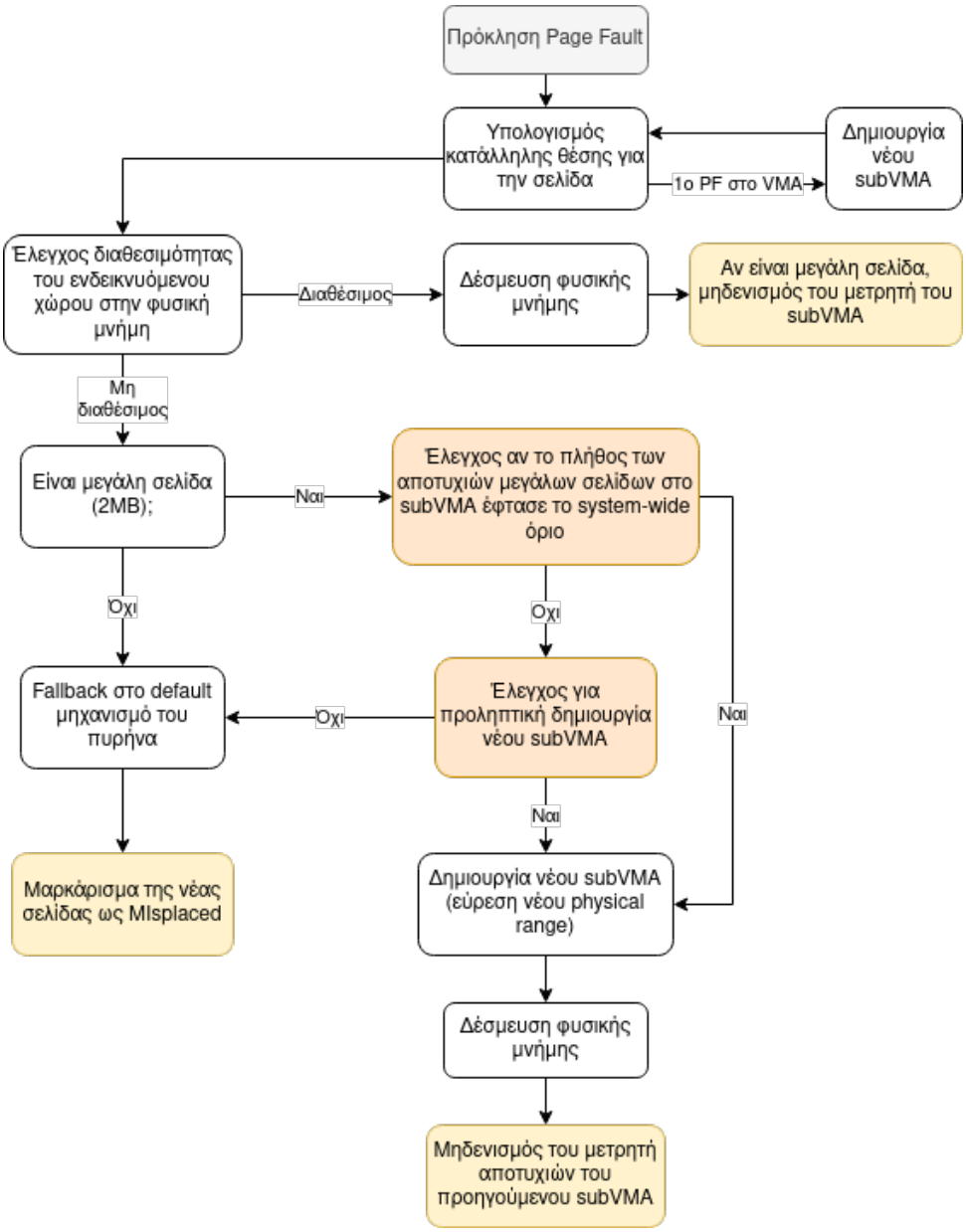
Ο μηχανισμός ¨ενεργοποιείται¨ όταν αποτύχει η τοποθέτηση μίας μεγάλης σελίδας (που ανήκει σε VMA), ο αριθμός των αποτυχημένων τοποθετήσεων (στο ίδιο subVMA) δεν έχει φτάσει το σχετικό όριο και είναι δυνατή η προσθήκη νέου subVMA.

Αναλυτικά ο μηχανισμός:

- Ξεκινώντας από το block σελίδων στο οποία απέτυχε να μπει η μεγάλη σελίδα, ελέγχει σειριακά τα επόμενα 2MB blocks της φυσικής μνήμης μέχρι ενός συγκεκριμένου πλήθους blocks.

- Το πλήθος αυτό καθορίζεται από το system-wide όριο αποτυχημένων τοποθετήσεων μεγάλων σελίδων μείον την τρέχουσα τιμή του σχετικού μετρητή του subVMA. Αν το πλήθος αυτό είναι μεγαλύτερο από αυτό που απομένει στο subVMA μέχρι το επόμενο subVMA στην φυσική μνήμη, τότε τερματίζει πρόωρα και δεν δημιουργεί νέο subVMA.

- Για κάθε block ελέγχει αν είναι ελεύθερο, κοιτώντας στις buddy λίστες. Αν έστω και ένα block το βρει διαθέσιμο για την τοποθέτηση μεγάλης σελίδας, τότε τερματίζει και δεν δημιουργείται νέο subVMA. Με άλλα λόγια, για να δημιουργήσει πρόωρα ένα νέο subVMA, πρέπει να βρει όλο το εύρος που θα ελέγξει μη διαθέσιμο για μεγάλες σελίδες.



Σχήμα 4.1: *Διάγραμμα ροής τροποποιημένου CaPaging*

4.6 Μετακινήσεις σελίδων

Η συνεισφορά του Translation Ranger στον τελικό μηχανισμό είναι η πραγματοποίηση των αναγκαίων μετακινήσεων σελίδων ασύγχρονα, ώστε να φέρνει στην ενδεικνυόμενη θέση την σελίδα σε επόμενο χρόνο από το allocation. Τα migrations γίνονται από ένα νήμα του πυρήνα που τρέχει περιοδικά. Το νήμα, διατρέχοντας τα VMAs, ελέγχει όλο τον χώρο εικονικών διευθύνσεων της διεργασίας και για κάθε εικονική σελίδα που βρίσκεται στην μνήμη βρίσκει το subVMA στο οποίο ανήκει. Αν δεν είναι στην ενδεδειγμένη θέση, προσπαθεί να μετακινήσει την σελίδα στην σωστή θέση. Το νήμα έχει την δυνατότητα να μετακινήσει ανώνυμες και file-backed σελίδες μεγέθους 4KB ή 2MB.

4.7 Τροποποίηση του Buddy Allocator

Ένα από τα ζητήματα που παρατηρήθηκαν στην λειτουργία ενός αρχικού συνδυασμού των δύο μηχανισμών είναι ότι αρκετές σελίδες είχαν τοποθετηθεί λανθασμένα επειδή στην υποδεικνυόμενη θέση ήταν μη-μετακινήσιμες σελίδες. Μάλιστα, δεν το παρατηρήσαμε λίγες τυχαίες φορές κατά την εκτέλεση ενός πειράματος, αλλά σε κάθε εκτέλεση του νήματος του Translation Ranger, κατά την διάρκεια εκτέλεσης του προγράμματος αδυνατούσε συστηματικά να μετακινήσει συγκεκριμένες σελίδες. Αυτές ήταν σελίδες πυρήνα, τις οποίες ο CaPaging άφησε με σκοπό να μετακινηθούν σε επόμενο χρόνο. Θα μπορούσαμε να είχαμε σχεδιάσει τον νέο τροποποιημένο CaPaging με τρόπο που στην περίπτωση τέτοιων σελίδων να μην τις αγνοούσε, αλλά να δημιουργούσε νέο subVMA. Όμως, αυτό δεν θα άλλαζε τίποτα, γιατί ο κατακερματισμός εξαιτίας τους θα παρέμενε.

Το πρόβλημα έπαιρνε σημαντική διάσταση, όταν αυτές ήταν απλωμένες στον χώρο της φυσικής μνήμης. Για τον περιορισμό αυτού και την συγκέντρωση αυτών των σελίδων μακριά από τα εύρη φυσικής μνήμης που είχε επιλέξει ο CaPaging, τροποποιήσαμε κατάλληλα τον αλγόριθμο του buddy allocator. Εκμεταλλευόμενοι το γεγονός ότι η λίστα μεγαλύτερης τάξης (MAX ORDER λίστα) είναι ταξινομημένη, η αλλαγή που υλοποιήσαμε είναι να προσθέτει και να αφαιρεί στις λίστες του τα στοιχεία από το τέλος. Ώστε, όταν μία σελίδα δεσμεύεται από άλλη διεργασία ή τον πυρήνα και χρειαστεί να σπάσει ένα μπλοκ μέγιστης τάξης, αυτό το μπλοκ να βρίσκεται, όποτε γίνεται, προς το τέλος της φυσικής μνήμης. Με αυτόν τον τρόπο, δεν αλλάζει η δέσμευση σελίδων από το CaPaging, η οποία συνήθως ξεκινά από σελίδες χαμηλά (μικρό PFN) στην φυσική μνήμη, λόγω ότι ο δείκτης κινείται προς τα δεξιά στον contiguity map, και συγχρόνως επικεντρώνει τις δεσμεύσεις σελίδων εκτός CaPaging στο άλλο άκρο της φυσικής μνήμης.

4.8 Παραμετροποιησιμότητα μηχανισμού

Ο νέος μηχανισμός έχει αρκετές παραμέτρους και επιλογές για το πως θα λειτουργήσει. Αρκετές δυνατότητες του μηχανισμού, όπως το μαρκάρισμα των σελίδων, το όριο των ανεκτών αποτυχιών μεγάλων σελίδων και η προληπτική δημιουργία νέων subVMAs, ελέγχονται μέσω επιλογών του sysctl.

Κεφάλαιο 5

Αξιολόγηση

Στο κεφάλαιο αυτό, θα παρουσιάσουμε τα αποτελέσματα που λάβαμε από τα πειράματα με τους τρεις μηχανισμούς (CaPaging, MACAP, TRanger). Επίσης, τα πειράματα δεν περιορίζονται στις μετρικές κάλυψης της φυσικής μνήμης (range coverage), αλλά θα δείξουμε και την συμπεριφορά των μηχανισμών όσον αφορά τις μετακινήσεις σελίδων (MACAP, TRanger), τις επιτυχημένες αρχικές τοποθετήσεις σελίδων (CaPaging, MACAP) και τέλος τους χρόνους εκτέλεσης.

Αρχικά, θα περιγράψουμε το περιβάλλον εκτέλεσης (χαρακτηριστικά μηχανημάτων, εκδόσεις πυρήνα, κ.λ.π.) και τα benchmarks με τα οποία αξιολογήσουμε τον νέο μηχανισμό μας. Επίσης, θα αναφερθούμε συνοπτικά στο πως συλλέξαμε τα δεδομένα για την εξαγωγή των διαφόρων μετρικών.

Στην συνέχεια, θα παρουσιάσουμε την συμπεριφορά των μηχανισμών σε μηχάνημα που μόλις έχει φορτώσει, δηλαδή μηχάνημα με τον μικρότερο δυνατό κατακερματισμό (επί της ουσίας μηδενικό). Ασχολούμαστε και με αυτό το σενάριο για να δείξουμε αν ο νέος μηχανισμός έχει καλές επιδόσεις και στις περιπτώσεις στις οποίες ήδη μας καλύπτουν οι TRanger και CaPaging. Στην συνέχεια, θα παραθέσουμε αποτελέσματα από διάφορα σενάρια κατακερματισμένης μνήμης. Ομοίως και εδώ, θα παρουσιαστούν τα αποτελέσματα με τρόπο, όπου είναι εύκολη και ξεκάθαρη η σύγκριση των μηχανισμών.

Τέλος, θα αναλύσουμε συνολικά την συμπεριφορά του νέου μας μηχανισμού. Θα επιδιώξουμε μία πληρέστερη σύγκριση με τους CaPaging και TRanger ξεχωριστά, ώστε να φανεί που υπερέχει και που όχι ο νέος μηχανισμός.

Πριν προχωρήσουμε στην παρουσία της μεθοδολογίας και των αποτελεσμάτων, είναι σημαντικό να αναφέρουμε ότι για τα πειράματα στον μηχανισμό MACAP έχουμε επιλέξει το *500 ως κατώφλι των συνεχόμενων αποτυχημένων τοποθέτησεων για μεγάλες σελίδες*. Το όριο αυτό το επιλέξαμε μετά από κάποιες γρήγορες αρχικές δοκιμές, διότι είδαμε ότι μόνο για τιμές από 500 και πάνω περιοριζόταν ο αριθμός των subVMAs που δημιουργούνταν. Σίγουρα, θα μπορούσε να αλλάζει δυναμικά το κατώφλι, αλλά αυτό αφήνεται για μελλοντική εργασία πάνω στον MACAP.

5.1 Μεθοδολογία

Για την αξιολόγηση του μηχανισμού, που περιγράψαμε εκτενώς στο πρηγούμενο κεφάλαιο, εκτελέσαμε κάποια πειράματα χρησιμοποιώντας τον. Η εκτέλεση των πειραμάτων πραγματοποιήθηκε σε ένα εικονικό μηχάνημα αρχτεκτονικής x86-64, στο οποίο αρχικά εγκαταστήσαμε μία διανομή Linux και το λειτουργούσαμε με έναν τροποποιημένο πυρήνα με τον μηχανισμό MACAP.

Επιλέξαμε τον πυρήνα Linux έκδοσης 4.19.160 ως βάση για το στήσιμο του μηχανισμού μας. Επιλέχθηκε αυτή η έκδοση, διότι η 4.19 είναι LTS (έκδοση Long Term Support). Επιπλέον, ο CaPaging υλοποιήθηκε σε πυρήνα έκδοσης 4.19.X και θεωρήσαμε πιο εύκολη την προσαρμογή του TRanger σε αυτήν την έκδοση, για τον οποίο είχαμε μόνο διαθέσιμες υλοποιήσεις στις εκδόσεις 4.16 και 5.0.

Στις επόμενες υποενότητες, θα αναφέρουμε αναλυτικά τα στοιχειά του φυσικού και εικονικού μηχανήματος, τα πειράματα (benchmarks) που πραγματοποιήσαμε και την μεθοδολογία συλλογής μετρικών.

5.1.1 Περιβάλλον εκτέλεσης

Χαρακτηριστικά φυσικού μηχανήματος

Για τις ανάγκες της εργασίας, έγινε χρήση ενός φυσικού μηχανήματος του εργαστηρίου Υπολογιστικών Συστημάτων (CSLab). Το μηχάνημα αυτό έχει τα εξής χαρακτηριστικά:

Κόμβοι NUMA	2
Επεξεργαστής	2x Intel Xeon CPU E5-2630, 2 threads/core, 10 cores, 2.2GHz
Συνολική Φυσική μνήμη	256GB
Διανομή Linux	Debian 11
Έκδοση πυρήνα	5.10.0-8-amd64
Εκδοση QEMU	5.2.0

Πίνακας 5.1: *Χαρακτηριστικά φυσικού μηχανήματος*

Κόμβοι NUMA	1
Επεξεργαστής	Intel Xeon CPU E5-2630, 16 vCPUs, 2.2GHz
Συνολική Φυσική μνήμη	200GB
Διανομή Linux	Debian 10.7.0
Έκδοση πυρήνα	custom 4.19.160

Πίνακας 5.2: *Χαρακτηριστικά εικονικού μηχανήματος*

Χαρακτηριστικά εικονικού μηχανήματος

Το εικονικό μηχανήμα το διαχειρίζεται ο hypervisor QEMU, υποστηριζόμενος από τον μηχανισμό KVM του πυρήνα του φυσικού μηχανήματος. Το KVM είναι μία τεχνολογία εικονικοποίησης ενσωματωμένη στον πυρήνα του Linux. Ο συνδυασμός QEMU+KVM μειώνει σημαντικά το overhead της εικονικοποίησης κατά την λειτουργία του VM.

Το QEMU προσφέρει στο εικονικό μηχάνημα έναν εικονικό επεξεργαστή ίδιων χαρακτηριστικών με αυτό του φυσικού μηχανήματος, αλλά επιλέξαμε να έχει λιγότερες εικονικές διαθέσιμες υπολογιστικές μονάδες (16 vCPUs). Για να έχουμε μόνο ένα κόμβο NUMA, όλες οι vCPUs ανήκουν στον ίδιο εικονική κόμβο. Επίσης στο εικονικό μηχάνημα διατίθενται και 200GB φυσικής μνήμης.

Τέλος, για το εικονικό μηχάνημα επιλέξαμε ένα Linux distro και συγκεκριμένα, το Debian Wheezy 10.7, το οποίο έρχεται εξ αρχής με πυρήνα έκδοσης 4.19.x, δηλαδή ίδιο με αυτό στο οποίο προσαρμόσαμε τον μηχανισμό μας.

5.1.2 Benchmarks

Για την αξιολόγηση του μηχανισμού μας και την διερεύνηση της συμπεριφοράς των CaPaging, TRanger, χρησιμοποιήσαμε 4 benchmarks:

1. **Micro** : Πρόκειται για ένα microbenchmark που σχεδιάσαμε για τον γρήγορο έλεγχο του μηχανισμού κατά την υλοποίησή του. Έχει σχετικά μικρή διάρκεια και τρέχει σειριακά. Το πρόγραμμα αυτό δεσμεύει συνολικά 120GB (σε ανώνυμα mappings) σε 3 φάσεις. Στην πρώτη φάση δεσμεύει το 10% των συνολικών, στην δεύτερη το 80% και στην τρίτη το υπόλοιπο 10%. Μετά από κάθε φάση προκαλεί τα κατάλληλα page faults για να δεσμευτούν σελίδες στην φυσική μνήμη και κάνει μερικούς άχρηστους υπολογισμούς για να μην τερματίσει γρήγορα.

2. **Liblinear** : Πρόκειται για μία βιβλιοθήκη προγραμμάτων για μεγάλης κλίμακας γραμμική ταξινόμηση [9]. Τρέχει σειριακά και για μεγάλα dataset (όπως αυτό στην περίπτωση μας) διαρκεί αρκετό χρόνο (περίπου 15 λεπτά). Λόγω του μικρού αποτύπωματός του στην φυσική μνήμη (40-45GB) έχει καλά αποτελέσματα με όλους τους μηχανισμούς σε όλα τα σενάρια εκτέλεσης. Για αυτό το λόγο, δεν το χρησιμοποιούμε κυρίως για να δείξουμε βελτίωση της κάλυψης με τον νέο μηχανισμό, αλλά λόγω μίας πολύ χρήσιμης ιδιαιτερότητάς του: το liblinear σε σχέση με τα άλλα benchmarks κάνει έντονη χρήση της PageCache (το 1/3 περίπου του συνολικού του αποτυπώματος).

3. **XSBench** : Πρόκειται για ένα πρόγραμμα μοντελοποίησης ,[10], που χρειάζεται αρκετή φυσική μνήμη (120GB) και έχει μικρή χρονική διάρκεια, αν και αυτό είναι μεταβλητό ανάλογα το σενάριο εκτέλεσής του. Το πρόγραμμα χωρίζεται σε δύο φάσεις. Η πρώτη αφορά την αρχικοποίηση των δεδομένων (πχ. πίνακες), κατά την οποία δεσμεύεται σχεδόν όλη η φυσική μνήμη που θα χρειαστεί. Η δεύτερη αφορά τους υπολογισμούς που γίνονται στα δεδομένα. Η πρώτη φάση εκτελείται σειριακά και διαρκεί πολύ περισσότερο από την δεύτερη, ενώ η δεύτερη εκτελείται με χρήση OpenMP threads και διαρκεί πολύ λίγο. Για τα πειράματά μας, επιλέξαμε το XSBench να δημιουργεί 10 νήματα.

4. **Hashjoin** : Πρόκειται για ένα microbenchmark. Και αυτό αποτελείται από δύο φάσεις. Στην πρώτη φάση αρχικοποιεί ένα μεγάλο τυχαίο hash table και στην δεύτερη κάνει αναζητήσεις (lookup) σε αυτήν την δομή. Έχει μικρή σχετικά διάρκεια (στα δικά μας σενάρια εκτέλεσης). Επίσης, όπως και το XSBench, έχει την δυνατότητα να παραλληλοποιήσει την δεύτερη φάση (τα lookups), αλλά όχι την πρώτη φάση των μαζικών

δεσμεύσεων η οποία τρέχει σειριακά. Στα πειράματα μας, για να έχουμε άλλο ένα benchmark που τρέχει παράλληλα, δίνουμε την δυνατότητα στο Hashjoin να χρησιμοποιεί 10 νήματα OpenMP για την φάση των lookups.

Επίσης, στον πίνακα 5.3, φαίνεται το μέγεθος της μνήμης που δεσμεύει περίπου κάθε benchmark και το μέγιστο πλήθος των VMAs που δημιουργούν. Στο Liblinear ο πρώτος αριθμός είναι αυτός της φυσικής μνήμης που έχει δεσμευτεί για VMAs και ο δεύτερος για την PageCache.

Benchmark	**Σύνολο φυσικής μνήμης που δεσμεύτηκε**	**Μέγιστο #VMAs**
Micro	~120GB	23
XSBench	~116.5GB	62
Hashjoin	~116.6GB	57
Liblinear	~29.1GB+~14GB(PageCache)	41

Πίνακας 5.3: *Μέγεθος αποτυπώματος στην μνήμη και μέγιστο πλήθος VMAs ανά benchmark.*

5.1.3 Μετρικές αξιολόγησης

Η αξιολόγηση του νέου μηχανισμού θα γίνει μέσω συγκεκριμένων μετρικών της συμπεριφοράς των μηχανισμών. Αυτές οι μετρικές αφορούν τρία ζητήματα, την απόδοσή τους όσον αφορά την μεταφραστική συνέχεια, διάφορες ενέργειές τους (μετακινήσεις, δεσμεύσεις) και τον χρόνο εκτέλεσής τους. Οι κύριες μετρικές είναι αυτές σχετικά με τα δύο πρώτα ζητήματα, διότι ο χρόνος εκτέλεσης δεν αποτυπώνει τα οφέλη της χρήσης των μηχανισμών στις κατάλληλες υλοποιήσεις υλικού (π.χ. range TLB), αλλά σε μηχανήματα με την κλασσική κυρίαρχη αρχιτεκτονική. Παρ' όλο αυτό, είναι χρήσιμες μετρικές για να αναδείξουν την ύπαρξη λειτουργιών στους μηχανισμούς που προσθέτουν χρονικές επιβαρύνσεις (π.χ. κοστοβόρες αναζητήσεις σε μεταδεδομένα ή περιττές ενέργειες).

5.1.4 Διαδικασία συλλογής αποτελεσμάτων

Για την συλλογή αποτελεσμάτων και την κατάλληλης επεξεργασία τους επεκτείναμε τα εργαλεία και τα scripts που είναι ανοιχτά διαθέσιμα για τους μηχανισμούς TRanger και CaPaging. Επίσης, για να βεβαιωθούμε ότι οι τιμές που λάβαμε είναι έγκυρες, επαναλάβαμε 4-5 φορές κάθε εκτέλεση πειράματος.

Για την μέτρηση της κάλυψης και την εξαγωγή των σχετικών μετρικών χρησιμοποιήσαμε το ήδη τροποποιημένο *pagecollect*[1] του CaPaging. Ο μηχανισμός αυτός εκτελείται ανά κάποια χρονική περίοδο και μέσω του pagemap διατρέχει τον εικονικό χώρο διευθύνσεων μίας διεργασίας και εξάγει αποτελέσματα ελέγχοντας για την ύπαρξη ευρών συνεχόμενων αντιστοιχίσεων (αυτά που εκμεταλλεύεται και το range TLB). Όσον αφορά την διαχείριση της εκτέλεσης των πειραμάτων (run scripts), αυτή βασίστηκε σε μία αρκετά τροποποιημένη έκδοση των αντίστοιχων εργαλείων[2] του TRanger. Οι υπόλοιπες μετρικές τις αποθηκεύσαμε στο τέλος της εκτέλεσης και διαβάστηκαν από διάφορα υποσυστήματα του πυρήνα. Τα

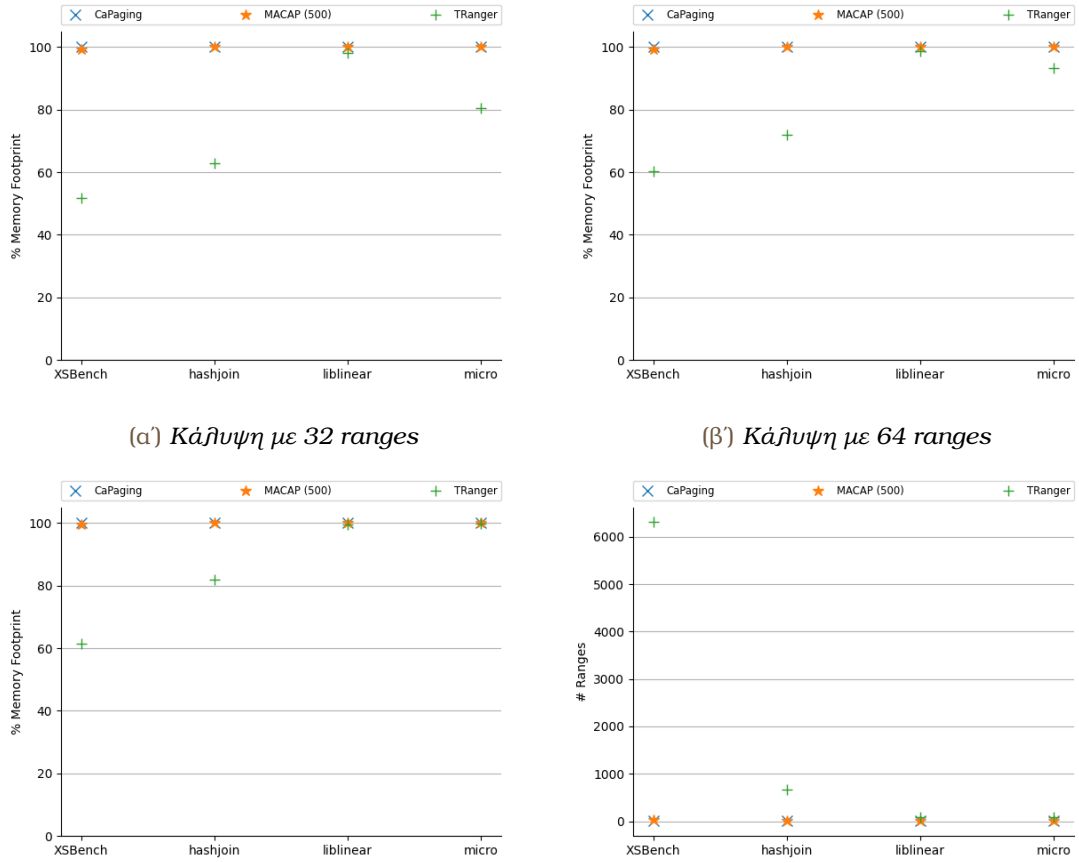
[1] https://github.com/cslab-ntua/contiguity-isca2020
[2] https://github.com/ysarch-lab/translation_ranger_userspace

εργαλεία και scripts που χρησιμοποιήθηκαν είναι ανοιχτά προσβάσιμα στην πλατφόρμα του GitHub[3].

Οι τελικές τιμές, για τις μετρικές που παρουσιάζουμε στην συνέχεια, προκύπτουν από την μέση τιμή των σχετικών τιμών από όλες τις σχετικές επαναλήψεις. Η μόνη εξαίρεση σε αυτό είναι οι χρόνοι εκτέλεσης. Για να ταιριάζουν μεταξύ τους οι τιμές των διάφορων τύπων χρόνου(π.χ. vCPU time ίση με User time + System time), κινηθήκαμε ως εξής: ταξινομήσαμε τις εκτελέσεις βάσει του συνολικού χρόνου εκτέλεσης (Real time) και κρατήσαμε τους χρόνους της εκτέλεσης που είναι στην μέση της ταξινομημένης λίστας των εκτελέσεων.

5.2 Πειράματα μετά από εκκίνηση μηχανήματος

Σε αυτήν την υποενότητα, όπως αναφέραμε, θα συγκρίνουμε τον MACAP με τους άλλους δύο μηχανισμούς για να δείξουμε αν ο νέος μηχανισμός παρουσιάζει χειρότερη ή καλύτερη επίδοση από τους επιμέρους μηχανισμούς στο σενάριο καθαρής μη-κατακερματισμένης μνήμης, στο οποίο ειδικά ο CaPaging παρουσιάζει άριστη επίδοση.



(α΄) *Κάλυψη με 32 ranges*

(β΄) *Κάλυψη με 64 ranges*

(γ΄) *Κάλυψη με 128 ranges*

(δ΄) *Αριθμός ranges για το 99% κάλυψη*

Σχήμα 5.1: *Γραφήματα κάλυψης μνήμης σε καθαρή μη-κατακερματισμένη μνήμη*

[3] https://github.com/c0louri/macap-tools

5.2.1 Κάλυψη μνήμης

Παρατηρώντας τα διαγράμματα (Σχήμα 5.1) για την κάλυψη μνήμης, φαίνεται ότι ο νέος μηχανισμός παρουσιάζει πολύ καλή επίδοση. Η κάλυψη μνήμης που προσφέρει είναι στα ίδια επίπεδα με το CaPaging. Ακόμα και στην μετρική του αριθμού των ευρών για την κάλυψη του 99% της μνήμης δεν υπολείπεται καθόλου του CaPaging. Μάλιστα, αυτή η εικόνα είναι κοινή και στα 4 benchmarks.

Σε σχέση με το TRanger έχει σαφώς καλύτερα αποτελέσματα, το οποίο ήταν αναμενόμενο, διότι ο νέος μηχανισμός δεσμεύει αρχικά τις σελίδες όπως ο CaPaging. Οπότε, λόγω της μη-κατακερματισμένης μνήμης έχει αρκετά λίγες αποτυχίες στην αρχική τοποθέτηση των νέων σελίδων.

Τα μόνο benchmarks στα οποία η επίδοση του TRanger είναι παρόμοια με αυτήν του MACAP είναι στα liblinear και micro.Το πρώτο διαρκεί αρκετό χρόνο, οπότε ο TRanger έχει αρκετές προσπάθειες για να πετύχει καλό αποτέλεσμα και το δεύτερο έχει την πιο απλή συμπεριφορά από άποψη του τι VMAs φτιάχνει. Στο micro όλα τα μεγάλα mappings που ζητά είναι ανώνυμα και επί της ουσίας ανήκουν στο ίδιο VMA, διότι κάθε καινούργιο mapping απλά προκαλεί την επέκταση ενός ήδη υπάρχοντος VMA στο τμήμα του σωρού στον χώρο εικονικών διευθύνσεων.

5.2.2 Δεσμεύσεις του CaP και μετακινήσεις σελίδων

Αρχικά θα σχολιάσουμε τις δεσμεύσεις του CaP, εννοώντας τις αρχικές προσπάθειες σωστής τοποθέτησης των σελίδων κατά την δέσμευσή τους. Εδώ μας ενδιαφέρουν κυρίως οι μεγάλες σελίδες, διότι με αυτές προσπαθούν κυρίως να καλύψουν τις ανάγκες της διεργασίας και οι δύο μηχανισμοί. Μόνη εξαίρεση σε αυτό είναι το Liblinear που χρησιμοποιεί αρκετές σελίδες βάσης. Πιθανότατα πρόκειται για τις σελίδες που δεσμεύτηκαν στο Page-Cache. Τέλος, όπως βλέπουμε στον πίνακα 5.4, το MACAP αφήνει λίγες παραπάνω μεγάλες σελίδες να αποτύχουν, το οποίο είναι αναμενόμενο, διότι τρέξαμε τα πειράματα σε μη-κατακερματισμένη μνήμη.

Benchmark	Μηχανισμός	4K επιτυχίες	2MB επιτυχίες
XSBench	CaPaging	760	59981
	MACAP	43	58377
Hashjoin	CaPaging	0	59677
	MACAP	0	58195
Liblinear	CaPaging	4257798	23420
	MACAP	4257303	22516
Micro	CaPaging	0	61412
	MACAP	146	61363

Πίνακας 5.4: *Πλήθος επιτυχημένων αρχικών τοποθετήσεων ανά μέγεθος σελίδας, σε καθαρή μνήμη.*

Μία ακόμα μετρική που παρακολουθήσαμε για να αξιολογήσουμε τον μηχανισμό μας είναι το πλήθος των μετακινήσεων σελίδων (μετρημένο σε 4K σελίδες) που πραγματοποιεί σε σχέση με το TRanger. Στον πίνακα 5.5 είναι φανερό ότι το νήμα του MACAP κάνει

πολύ λιγότερες μετακινήσεις, το οποίο ήταν αναμενόμενο διότι οι περισσότερες σελίδες είχαν τοποθετηθεί σωστά κατά την αρχική δέσμευσή τους.

Benchmark	**Μηχανισμός**	**Επιτυχημένες μετακινήσεις**	**Αποτυχημένες μετακινήσεις**
XSBench	TRanger	45438062	13626668
	MACAP	3093037	434383
Hashjoin	TRanger	110389506	8289159
	MACAP	2236017	85
Liblinear	TRanger	31515654	3083308
	MACAP	488613	78474
Micro	TRanger	88674057	465846
	MACAP	3150782	2320

Πίνακας 5.5: *Πλήθος μετακινήσεων σελίδων σε σενάριο καθαρής μνήμης*

5.2.3 Χρόνοι εκτέλεσης

Στον πίνακα 5.6, παραθέτουμε ενδεικτικούς χρόνους εκτέλεσης για τα πειράματα με τους τρεις μηχανισμός σε μη κατακερματισμένη μνήμη. Ο MACAP φαίνεται να μην προσθέτει σημαντική χρονική επιβάρυνση, και βλέπουμε σχεδόν σε όλα τα πειράματα να οδηγείται σε καλύτερους χρόνους από το TRanger και σε παρόμοιους με το CaPaging.

Benchmark	**Μηχανισμός**	**Real (ms)**	**User (ms)**	**System (ms)**
Micro	CaPaging	165106	119853	44725
	MACAP	174708	130615	44285
	TRanger	168590	126732	37267
XSBench	CaPaging	141672	278419	44828
	MACAP	144179	287527	44755
	TRanger	147131	296173	41600
Hashjoin	CaPaging	304911	2224597	42691
	MACAP	302941	2164965	43535
	TRanger	304699	2179074	46829
Liblinear	CaPaging	1037528	987520	40445
	MACAP	1032757	982599	41003
	TRanger	1039692	992234	38710

Πίνακας 5.6: *Μέσοι χρόνοι πειραμάτων σε κενή φυσική μνήμη*

Το μόνο benchmark που διαφοροποιείται από τα υπόλοιπα είναι το Micro. Σε αυτό φαίνεται ότι ο μηχανισμός MACAP, είναι ~6% πιο αργός από το CaPaging και ~4% πιο αργός από το TRanger. Η διάφορα οφείλεται κυρίως σε TLB misses, διότι η εκτέλεση με το MACAP καλύπτεται με πολύ λιγότερες μεγάλες σελίδες και ένα μεγάλο κομμάτι της δεσμευμένης μνήμης υποστηρίζεται από 4K σελίδες (Πίνακας 5.7). Όμως, αυτό το πρόβλημα δεν είναι τόσο σημαντικό, διότι γενικά μας ενδιαφέρει οι μηχανισμοί αυτοί να χρησιμοποιηθούν με range TLBs και σχετικές προτάσεις.

Τέλος, μία ακόμα παρατήρηση που αξίζει να αναλυθεί είναι ο χρόνος συστήματος, όπου στο TRanger είναι μικρότερος στις περισσότερες περιπτώσεις. Αυτό συμβαίνει για δύο λόγους.

Πρώτον, ο TRanger δεν δημιουργεί κάποια επιβάρυνση κατά την δέσμευση σελίδων, η οποία επιβάρυνση αποτυπώνεται στο χρόνο συστήματος της διεργασίας. Δεύτερον, η καθυστέρηση του TRanger, λόγω του νήματος που μετακινεί σελίδες, φαίνεται μόνο στον συνολικό χρόνο, διότι το νήμα λόγω κάποιου κλειδώματος τύπου semaphore βάζει σε αναμονή την διεργασία, μέχρι την απελευθέρωση του, και όταν μία διεργασία είναι σε αναμονή (στον χρονοδρομολογητή του λ.σ.) δεν δεσμεύει πόρους.

Benchmark	Μηχανισμός	Μεγάλες σελίδες	4K σελίδες
Micro	CaPaging	61437	1872
	MACAP	55292	3147859
	TRanger	61423	8783
XSBench	CaPaging	59674	2606
	MACAP	59640	19572
	TRanger	59638	20193
Hashjoin	CaPaging	59701	474
	MACAP	59700	752
	TRanger	59046	310490
Liblinear	CaPaging	14882	6930
	MACAP	14881	5732
	TRanger	14837	29564

Πίνακας 5.7: *Πλήθος σελίδων ανά μέγεθος σε μη κατακερματισμένη μνήμη.*

5.3 Πειράματα με κατακερματισμένη μνήμη

Στην ενότητα αυτή, παρουσιάζουμε και αναλύουμε τα αποτελέσματα που λάβαμε από την εκτέλεση πειραμάτων σε κατακερματισμένη μνήμη. Το ενδιαφέρον μας για την συμπεριφορά του μηχανισμού σε κατακερματισμένη μνήμη (όπου ένα αξιόλογο τμήμα της είναι ήδη υπό χρήση) προκύπτει από το γεγονός ότι αυτό ήταν το σενάριο που οδηγούσε σε χειρότερη απόδοση του CaPaging. Για αυτό και η αξιολόγηση επικεντρώνεται στην κάλυψη μνήμης σε κατακερματισμένη μνήμη. Όμως, δεν αδιαφορούμε για τις υπόλοιπες εκφάνσεις της εκτέλεσης, και για αυτό ερευνήσαμε την συνολικότερη συμπεριφορά του νέο μηχανισμού, όπως π.χ. οι μετακινήσεις σελίδων.

Για την πιο ολοκληρωμένη αξιολόγηση, εξετάζουμε τέσσερα διαφορετικά σενάρια κατακερματισμένης μνήμης που διαφέρουν στο ποσοστό της φυσικής μνήμης που παραμένει δεσμευμένη από άλλες διεργασίας του λ.σ. κατά την εκτέλεση των benchmarks. Τα αποτελέσματα που θα ακολουθήσουν προέκυψαν με 10%, 20%, 30% και 35% ήδη δεσμευμένης μνήμης για τα Micro, XSBench, Hashjoin, ενώ για το Liblinear, που δεσμεύει λιγότερη μνήμη, με 50%, 60%, 70% και 75% ήδη χρησιμοποιούμενης μνήμης.

Η δέσμευση συγκεκριμένου ποσοστού μνήμης και η δημιουργία κατακερματισμού έγινε από ένα εργαλείο-πρόγραμμα που εκτελείται πριν το benchmark. Το πρόγραμμα αυτό, αρχικά, δεσμεύει το σύνολο της φυσικής μνήμης ζητώντας μεγάλες σελίδες για τον φυσικό χώρο που θέλει να δεσμεύσει. Στην συνέχεια, ελευθερώνει σταδιακά τυχαία τμήματα (μεγέθους πολλαπλάσιου της μεγάλης σελίδας) από την μνήμη που έχει δεσμεύσει μέχρι να μην κατέχει πάνω από ένα ποσοστό της φυσικής μνήμης, που το ορίζουμε εμείς. Αφού ολο-

κληρωθεί αυτή η φάση αποδέσμευσης σελίδων, δεν τερματίζει και παραμένει ανοιχτό μέχρι το τέλος του πειράματος. Με αυτή την διαδικασία το εργαλείο αυτό δημιουργεί τον ζητούμενο κατακερματισμό στην μνήμη. Μάλιστα, επειδή πραγματοποιεί αποδεσμεύσεις τυχαία, τα ελεύθερα εύρη είναι πολλά και διάσπαρτα στον χώρο φυσικών διευθύνσεων, επιτυγχάνοντας έτσι μία κατάσταση φυσικής μνήμης που προσομοιώνει επιτυχώς το πως επηρεάζεται η φυσική μνήμη σε μηχάνημα που έχει τρέξει και τερματίσει πολλές διεργασίες.

Στην συνέχεια, ακολουθούν τα αποτελέσματα, τα οποία παρουσιάζονται με παρόμοια δομή με αυτήν που χρησιμοποιήσαμε στην προηγούμενη ενότητα.

5.3.1 Κάλυψη μνήμης

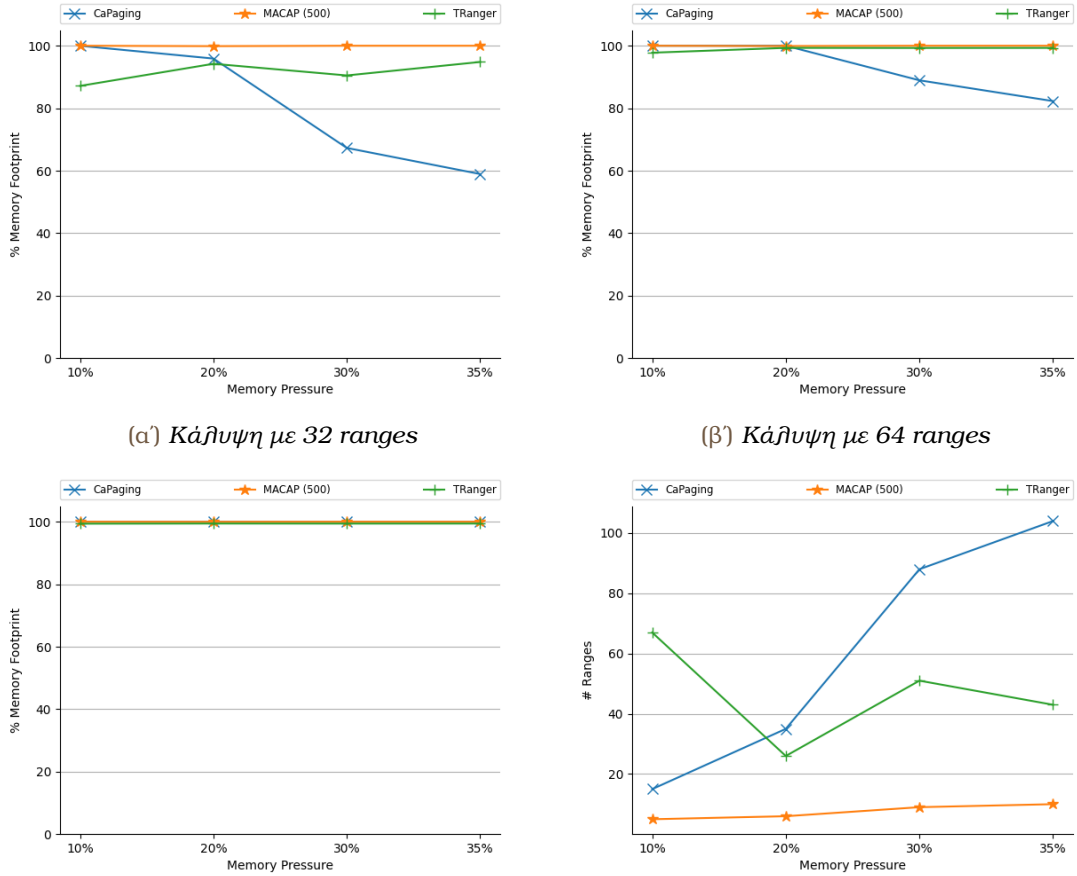
Οι μετρικές για την κάλυψη της μνήμης είναι αυτές που μας ενδιαφέρουν περισσότερο, αφού άλλωστε η επίτευξη μεταφραστικής συνέχειας είναι ο σκοπός για τον οποίο σχεδιάστηκαν όλοι οι μηχανισμοί που παρουσιάζονται στην διπλωματική εργασία. Βλέπουμε λοιπόν ότι ο νέος μηχανισμός παρουσιάζει εξαιρετικές επιδόσεις στον τομέα αυτό (Σχήματα 5.2 - 5.5). Στην συνέχεια, θα αναφερθούμε σε κάθε benchmark ξεχωριστά, και θα εξηγήσουμε τυχόν ενδιαφέρουσες παρατηρήσεις που είχαμε.

Στο **Micro**, η απόδοση του MACAP ήταν εξαιρετική σε κάθε περίπτωση (Σχήμα 5.2). Παρουσιάζει την καλύτερη εικόνα σε όλες τις μετρικές για την κάλυψη της μνήμης. Σε σχέση με τον CaPaging, το μέγεθος της ήδη χρησιμοποιημένης μνήμης δεν τον επηρεάζει καθόλου, ενώ ο TRanger έχει παρόμοια συμπεριφορά με το MACAP στο ζήτημα της διατήρησης της υψηλής μεταφραστικής συνέχειας. Όμως, ο MACAP τα καταφέρνει αρκετά καλύτερα στον αριθμό των ευρών για 99% κάλυψη σε σχέση και με τους άλλους δύο μηχανισμούς (Σχήμα 5.2δ').

Στο **Liblinear** δεν έχουμε κάτι άξιο αναφοράς. Και οι τρεις μηχανισμοί επιτυγχάνουν εξαιρετική κάλυψη (Σχήμα 5.3). Η μόνη διαφορά τους είναι στον αριθμό ευρών για 99% κάλυψη, όπου ο TRanger έχει σχετικά χειρότερα αποτελέσματα από τους άλλους δύο σε όλες τις περιπτώσεις πίεσης στην μνήμη (Σχήμα 5.3δ').

Στο **XSBench** ο μηχανισμός MACAP πετυχαίνει την καλύτερη μεταφραστική συνέχεια ανεξάρτητα από τον πόσο πιεσμένη ήταν ήδη η μνήμη (Σχήμα 5.4). Επίσης, στο συγκεκριμένο benchmark ο TRanger είχε την χειρότερη επίδοση από κάθε άλλο benchmark. Αυτό οφείλεται στο γεγονός ότι ο μηχανισμός δεν έχει αρκετό χρόνο για να μετακινήσει όλες τις σελίδες, αφού η φάση των υπολογισμών στο τέλος του προγράμματος διαρκεί αρκετά λίγο. Αντιθέτως, ο MACAP αν και πραγματοποιεί μετακινήσεις σελίδων, ανταπεξέρχεται μια χαρά και ο λόγος είναι ότι οι μετακινήσεις είναι αρκετά λιγότερες λόγω ότι μόνο οι μαρκαρισμένες σελίδες μετακινούνται, ενώ καμία σελίδα που έχει εξαρχής τοποθετηθεί σωστά δεν μετακινείται μετά.

Το benchmark **Hashjoin** είναι το μόνο στο οποίο δεν υπήρχε ένας μηχανισμός που να ήταν καλύτερος σε όλες τις μετρικές. Για τα μικρότερα ποσοστά (10%, 20%), καλύτερο αποτέλεσμα είχαμε με τον MACAP, ενώ στις άλλες περιπτώσεις (30%, 35%) ο TRanger πετύχαινε καλύτερα αποτελέσματα σε όλες τις μετρικές. Όμως, ο MACAP είχε καλή απόδοση με το Hashjoin σε όλες τις περιπτώσεις.



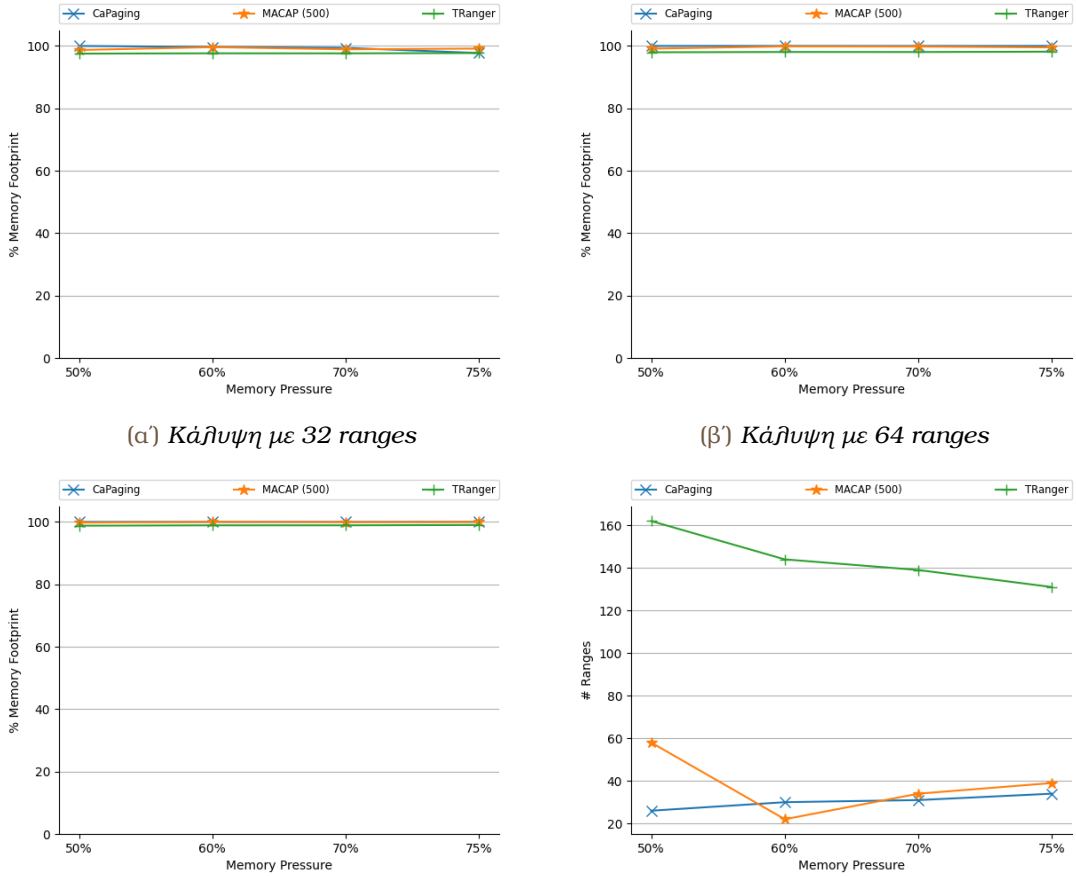
(α') *Κάλυψη με 32 ranges*

(β') *Κάλυψη με 64 ranges*

(γ') *Κάλυψη με 128 ranges*

(δ') *Αριθμός ranges για 99% κάλυψη*

Σχήμα 5.2: *Κάλυψη μνήμης για Micro.*



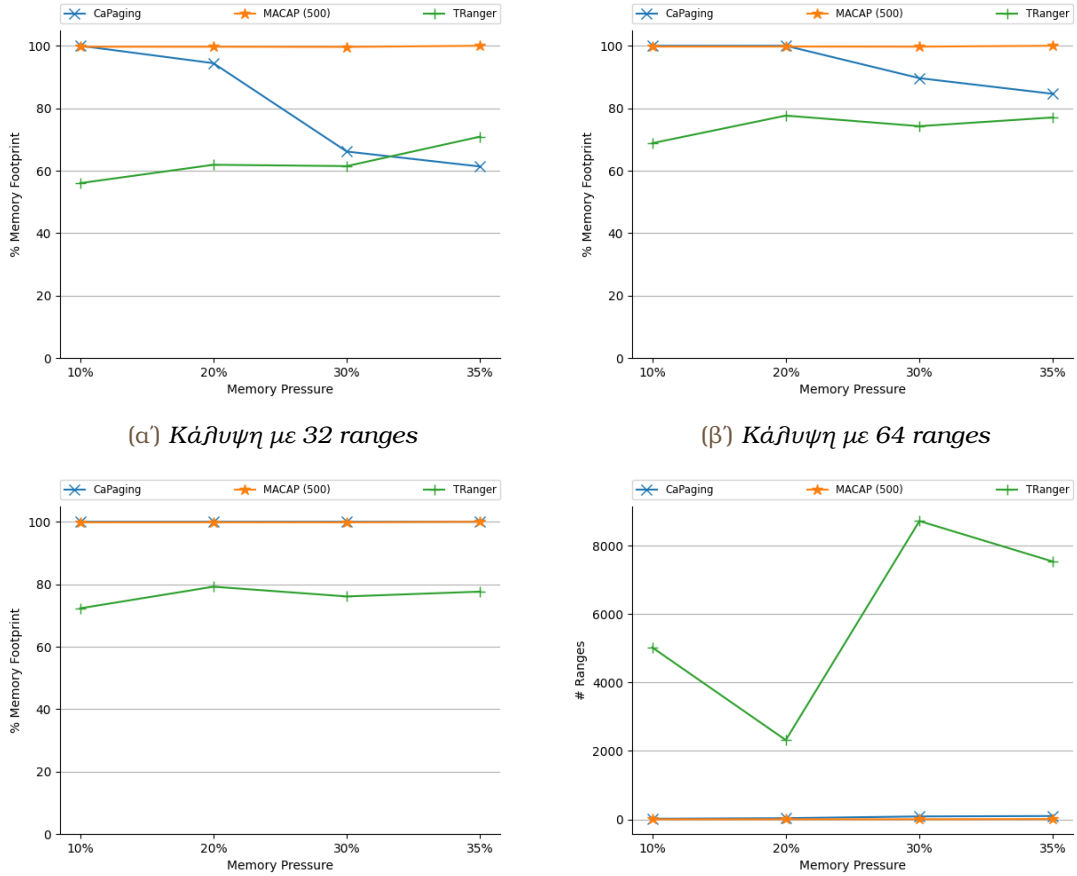
(α′) *Κάλυψη με 32 ranges*

(β′) *Κάλυψη με 64 ranges*

(γ′) *Κάλυψη με 128 ranges*

(δ′) *Αριθμός ranges για 99% κάλυψη*

Σχήμα 5.3: *Κάλυψη μνήμης με για Liblinear.*



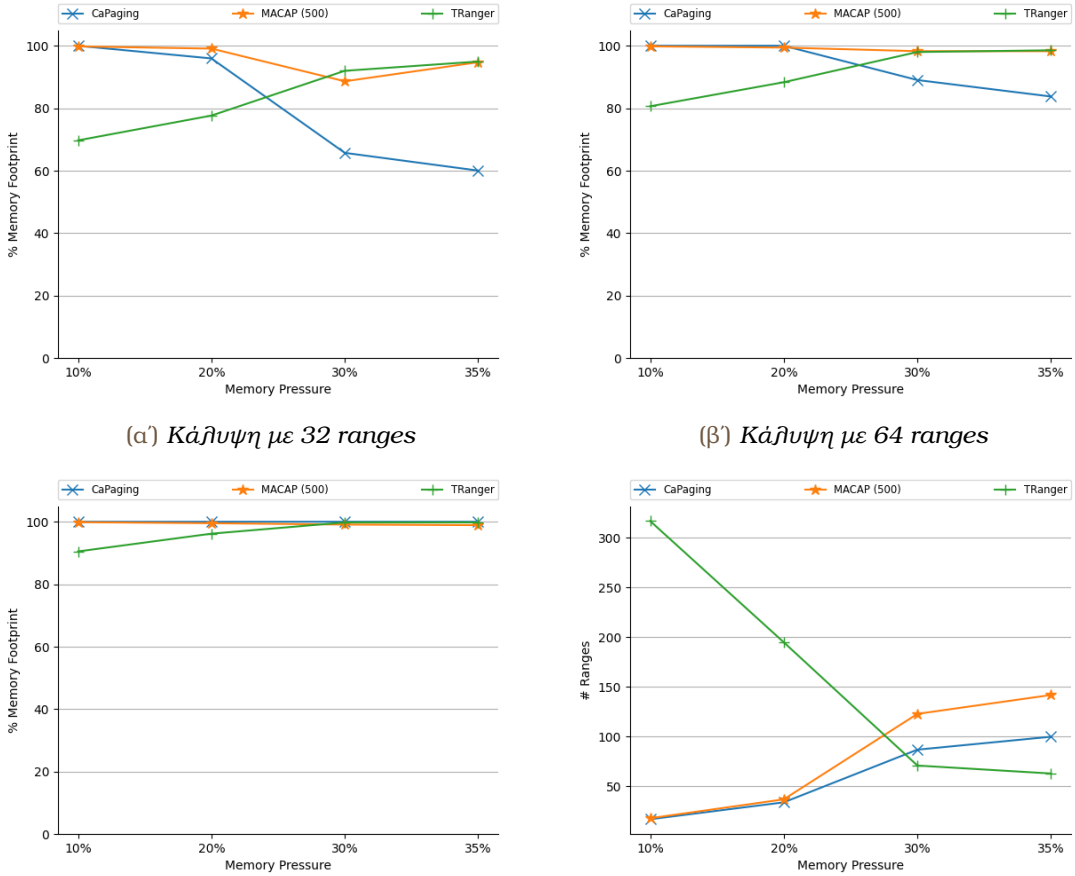
(α') *Κάλυψη με 32 ranges*

(β') *Κάλυψη με 64 ranges*

(γ') *Κάλυψη με 128 ranges*

(δ') *Αριθμός ranges για το 99% κάλυψη*

Σχήμα 5.4: *Κάλυψη μνήμης για XSBench.*



(α') *Κάλυψη με 32 ranges*

(β') *Κάλυψη με 64 ranges*

(γ') *Κάλυψη με 128 ranges*

(δ') *Αριθμός ranges για το 99% κάλυψη*

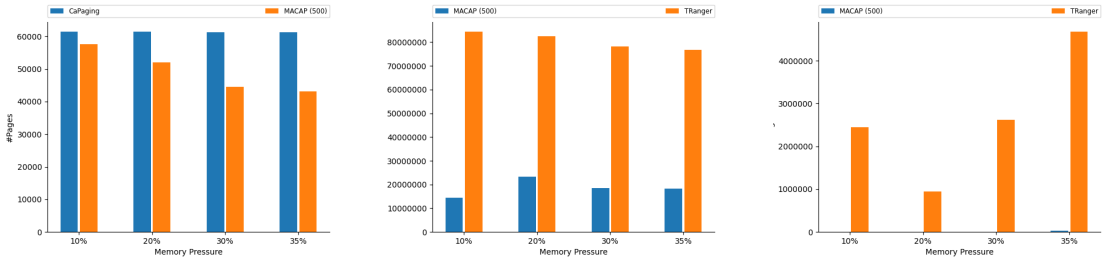
Σχήμα 5.5: *Κάλυψη μνήμης για Hashjoin.*

5.3.2 Δεσμεύσεις του CaP και μετακινήσεις σελίδων

Σε αυτήν την υποενότητα, θα ασχοληθούμε με τις μετρικές που αφορούν το πως μία σελίδα τοποθετήθηκε στην σωστή θέση. Με τον MACAP, οι σελίδες της διεργασίας μπορούν να τοποθετούνται στην υποδεικνυόμενη θέση είτε κατά την αρχική δέσμευση τους (δέσμευση CaP) ή αν αποτύχει η προηγούμενη προσπάθεια, να γίνει προσπάθεια τοποθέτησής τους σε επόμενο χρόνο από το νήμα του MACAP (μετακίνηση σελίδας). Μία από τις επιδιωκόμενες συμπεριφορές από τον MACAP είναι να θυσιάζει τις τοποθετήσεις κάποιων μεγάλων σελίδων που θα δημιουργούσαν νέο subVMA, και να τις μετακινεί σε επόμενο χρόνο μέσω του νήματος, ενώ συγχρόνως να αποφεύγει το μεγάλο πλήθος μετακινήσεων που κάνει ο TRanger.

Παρατηρώντας τα αποτελέσματα (Σχήματα 5.6, 5.7, 5.8, 5.9), βλέπουμε ότι ο νέος μηχανισμός έχει την ζητούμενη συμπεριφορά. Όσο αυξάνεται ο κατακερματισμός της μνήμης παρατηρούμε ότι μειώνονται οι επιτυχημένες τοποθετήσεις κατά την δέσμευση (αναμενόμενη εικόνα), ενώ συγχρόνως ,σε όλα τα benchmarks, το πλήθος των επιτυχημένων μετακινήσεων σελίδων είναι πολύ μικρότερο από αυτό με τον μηχανισμό TRanger. Επίσης, επιτυγχάνεται και ελαχιστοποίηση των αποτυχημένων μετακινήσεων.

Στην συνέχεια ακολουθούν τα σχετικά γραφήματα ομαδοποιημένα ανά benchmark και συγκεκριμένα τα γραφήματα των επιτυχημένων αρχικών 2M τοποθετήσεων και των επιτυχημένων και αποτυχημένων μετακινήσεων. Επίσης θεωρήσαμε περιττό να συμπεριλάβουμε και τα γραφήματα για την αρχική τοποθέτηση 4K σελίδων για δύο λόγους. Πρώτον, σχεδόν όλα τα page faults σε κάθε διεργασία είναι για μεγάλες σελίδες, με εξαίρεση το PageCache που χρησιμοποιείται για το Liblinear. Όμως, η διαδικασία δέσμευσης σελίδων για το PageCache δεν έχει αλλάξει (reserved lists στον CaPaging) και δεν επηρεάζεται από την λειτουργία του MACAP. Δεύτερον, η διαδικασία που ακολουθείται κατά την δέσμευση 4K σελίδων παραμένει ίδια στα CaPaging και MACAP. Αν και στο τέλος, υπάρχουν αρκετές αντιστοιχίσεις στην μνήμη με 4K σελίδες, το πλήθος των σελίδων βάσης έχει προκύψει κυρίως από διάσπαση μεγάλων σελίδων κατά την μετακίνησή τους από το νήμα και όχι από εξαρχής δεσμεύσεις 4K σελίδων.

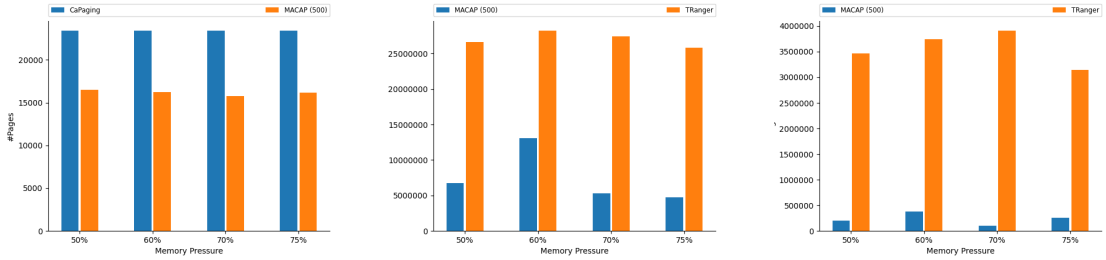


(α') *Επιτυχημένες αρχικές δεσμεύσεις*

(β') *Επιτυχημένες μετακινήσεις σελίδων (πλήθος σε 4K σελίδες)*

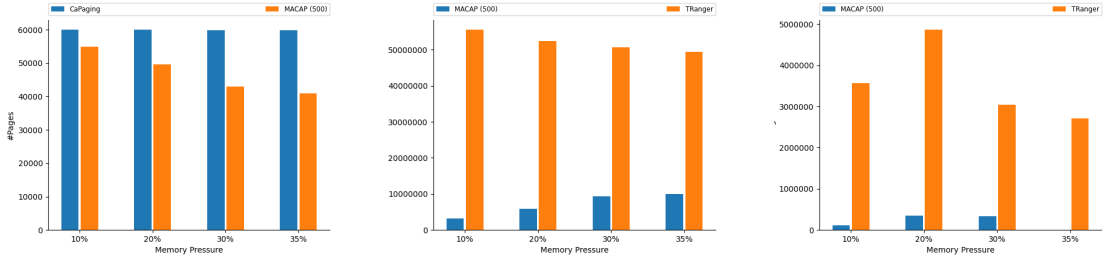
(γ') *Αποτυχημένες μετακινήσεις σελίδων (πλήθος σε 4K σελίδες)*

Σχήμα 5.6: *Δεσμεύσεις και μετακινήσεις μεγάλων σελίδων στο Micro.*



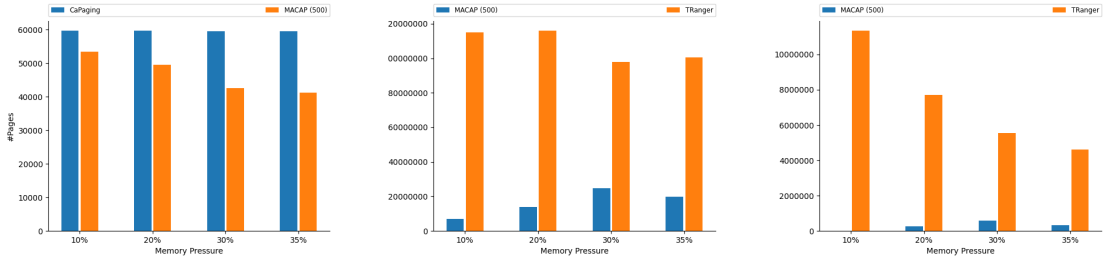
(α΄) *Επιτυχημένες αρχικές δεσμεύσεις* (β΄) *Επιτυχημένες μετακινήσεις σελίδων (πλήθος σε 4K σελίδες)* (γ΄) *Αποτυχημένες μετακινήσεις σελίδων (πλήθος σε 4K σελίδες)*

Σχήμα 5.7: *Δεσμεύσεις και μετακινήσεις μεγάλων σελίδων στο Liblinear.*



(α΄) *Επιτυχημένες αρχικές δεσμεύσεις* (β΄) *Επιτυχημένες μετακινήσεις σελίδων (πλήθος σε 4K σελίδες)* (γ΄) *Αποτυχημένες μετακινήσεις σελίδων (πλήθος σε 4K σελίδες)*

Σχήμα 5.8: *Δεσμεύσεις και μετακινήσεις μεγάλων σελίδων στο XSBench.*



(α΄) *Επιτυχημένες αρχικές δεσμεύσεις* (β΄) *Επιτυχημένες μετακινήσεις σελίδων (πλήθος σε 4K σελίδες)* (γ΄) *Αποτυχημένες μετακινήσεις σελίδων (πλήθος σε 4K σελίδες)*

Σχήμα 5.9: *Δεσμεύσεις και μετακινήσεις μεγάλων σελίδων στο Hashjoin.*

5.3.3 Χρόνοι εκτέλεσης

Στις προηγούμενες ενότητες, ασχοληθήκαμε με το πόσο καταφέρνει τον σκοπό του ο νέος μηχανισμός και πως το πραγματοποιεί. Με άλλα λόγια, προηγουμένως ασχοληθήκαμε με την συμπεριφορα του αλγορίθμου κατά την εκτέλεση των πειραμάτων. Όμως, δεν αρκούν μόνο αυτά. Όλοι αυτοί οι μηχανισμοί έχουν σχεδιαστεί με γνώμονα την μείωση της χρονικής επιβάρυνσης των μεταφράσεων διευθύνσεων, επομένως, πρέπει να παρατηρήσουμε και τους χρόνους εκτέλεσης των πειραμάτων για να δούμε τυχόν χρονικές επιβαρύνσεις λόγω λειτουργιών του μηχανισμού.

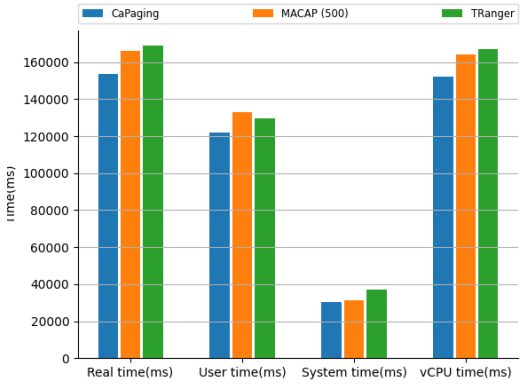
Στην ενότητα αυτή, μας ενδιαφέρει κυρίως να δούμε αν ο MACAP δημιουργεί χρονική επιβάρυνση που να κάνει ασύμφορη την χρήση του σε πραγματικές συνθήκες. Επίσης, λόγω ότι ο νέος μηχανισμός μετακινεί σελίδες, αλλά σε μικρότερο βαθμό από τον TRanger, περιμέναμε να έχει χρονική επίδοση ενδιάμεση των CaPaging και TRanger. Στην συνέχεια θα παρουσιάσουμε αναλυτικά την εικόνα σε κάθε benchmark.

Στο **Micro**, σε όλα τα σενάρια εκτέλεσης ο MACAP πετυχαίνει χρόνους ανάμεσα στους αντίστοιχους χρόνους των άλλων μηχανισμών (Σχήμα 5.10). Η επίδοση αυτή είναι αναμενόμενη, διότι περιμέναμε κάποια επιβάρυνση λόγω των μετακινήσεων σελίδων, αλλά όχι όση στον TRanger, ο οποίος πραγματοποιεί περισσότερες μετακινήσεις.

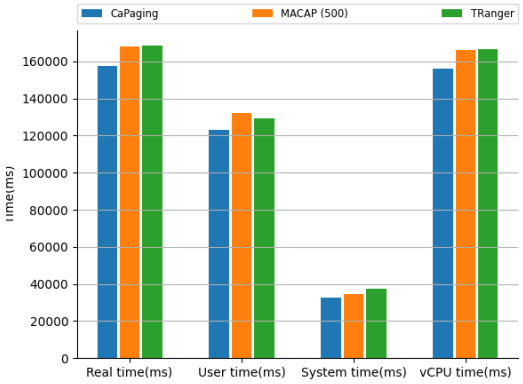
Στο **XSBench**, είχαμε επίσης παρόμοια συμπεριφορά με το Micro (Σχήμα 5.11). Όμως, σε αντίθεση με το Micro, σε αυτό οι εκτελέσεις με τον MACAP χρονικά είναι πιο κοντά σε αυτές με τον CaPaging. Η διαφορά οφείλεται στο γεγονός ότι ενώ οι ανάγκες σε μνήμη είναι παρόμοιες σε αυτά τα δυο benchmarks, στο XSBench ο MACAP κάνει αρκετά λιγότερες μετακινήσεις σελίδων σε σχέση με το πως ενέργησε στο Micro. Επιπλέον, πέρα από το ότι μετακινεί λιγότερες σελίδες, επίσης δημιουργεί πολύ λιγότερες αντιστοιχίσεις σελίδων βάσης. Ενδεικτικά, στην περίπτωση με 30% δεσμευμένη μνήμη, στο Micro καταλήγει να έχει 2446665 αντιστοιχίσεις 4κ σελίδων, ενώ στο XSBench 73701! Για αυτό και η σημαντική διαφορά των MACAP, CaPaging με τον TRanger στο User Time του XSBench.

Σε σχέση με τα προηγούμενα δύο benchmarks, στα **Liblinear** (Σχήμα 5.12) και **Hashjoin** (Σχήμα 5.13), δεν μπορούμε να εξάγουμε κάποιο ασφαλές συμπέρασμα σχετικά με το πιο benchmark υπερέχει. Δεν υπάρχει ομοιομορφία στην εικόνα που έχουμε από τα διάφορα σενάρια εκτέλεσης. Για παράδειγμα, στο Σχήμα 5.12γ΄ βλέπουμε ότι ξεκάθαρα γρηγορότερος είναι ο CaPaging και πιο αργός ο TRanger, ενώ στο Σχήμα 5.12α΄ ο CaPaging είναι ο πιο αργός και στο Σχήμα 5.12δ΄ ο TRanger έχει πετύχει τους καλύτερους χρόνους. Επίσης, στην περίπτωση του Hashjoin παρατηρήσαμε και μεγάλη διακύμανση στις διάφορες εκτελέσεις στο ίδιο σενάριο κατακερματισμού. Τουλάχιστον, από τα σχήματα σε αυτά τα δύο benchmarks μπορούμε να αναφέρουμε με σχετική ασφάλεια ότι ο MACAP δημιουργεί κατά μέσο όρο παρόμοια χρονική επιβάρυνση με αυτήν των άλλων μηχανισμών.

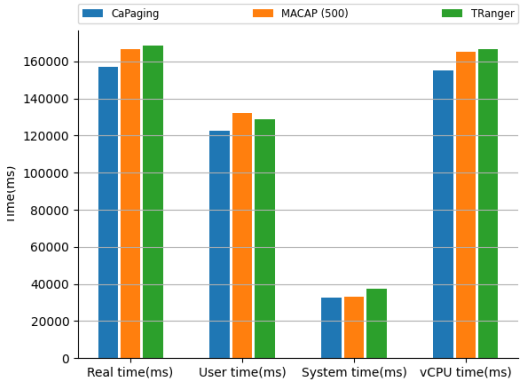
Τέλος, πρέπει να σημειωθεί ότι τα αποτελέσματα δεν είναι πλήρως αξιόπιστα για την εξαγωγή συμπερασμάτων. Αυτό οφείλεται, πρώτον, στο περιβάλλον εκτέλεσης, διότι εκτελούμε τα πειράματα σε εικονικό μηχάνημα και όχι κατευθείαν στο φυσικό. Επίσης, ένας δεύτερος λόγος είναι ότι γίνεται χρήση κλασσικού TLB, οπότε στις περιπτώσεις όπου ο MACAP έχει κατά 10%-20% καλύτερη κάλυψη (π.χ. στα 32 εύρη) από τους άλλους μηχανισμούς, αυτή η καλύτερη επίδοση δεν αποτυπώνεται στους χρόνους εκτέλεσης.



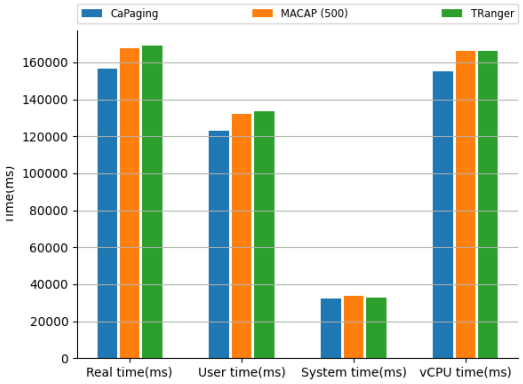
(α') *Με 10% ήδη δεσμευμένη μνήμη.*



(β') *Με 20% ήδη δεσμευμένη μνήμη.*

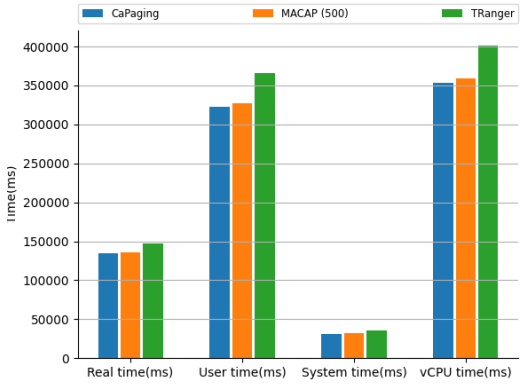


(γ') *Με 30% ήδη δεσμευμένη μνήμη.*

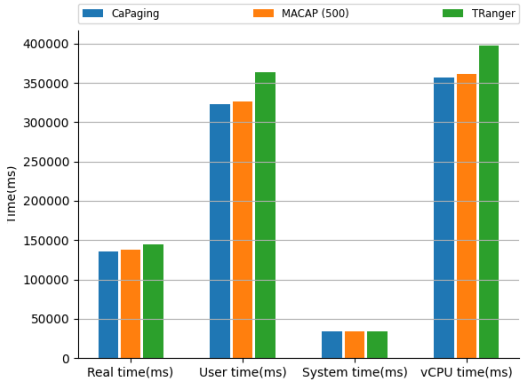


(δ') *Με 35% ήδη δεσμευμένη μνήμη.*

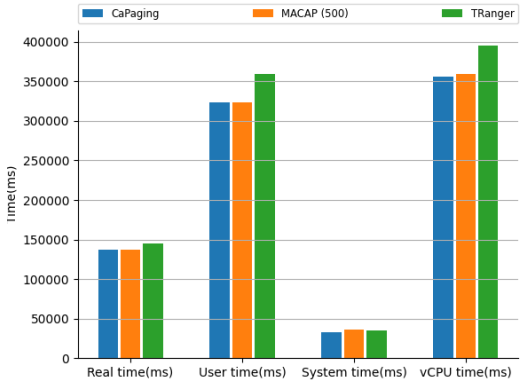
Σχήμα 5.10: *Χρόνοι εκτέλεσης για Micro.*



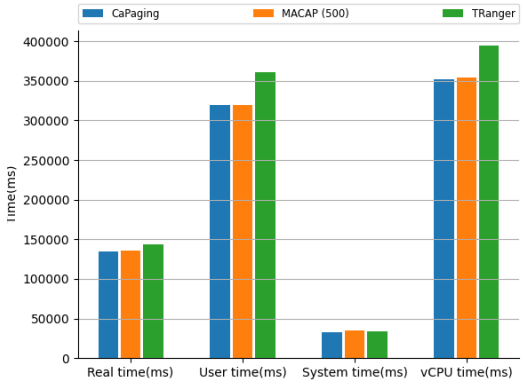
(α') *Με 10% ήδη δεσμευμένη μνήμη.*



(β') *Με 20% ήδη δεσμευμένη μνήμη.*

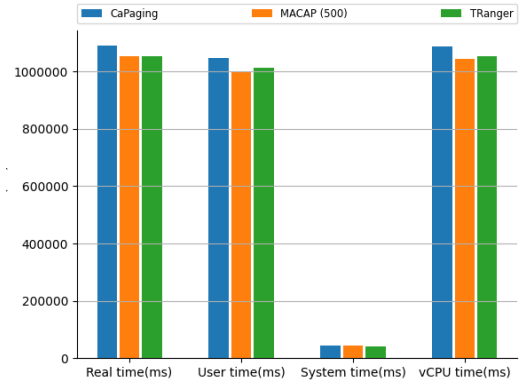


(γ') *Με 30% ήδη δεσμευμένη μνήμη.*

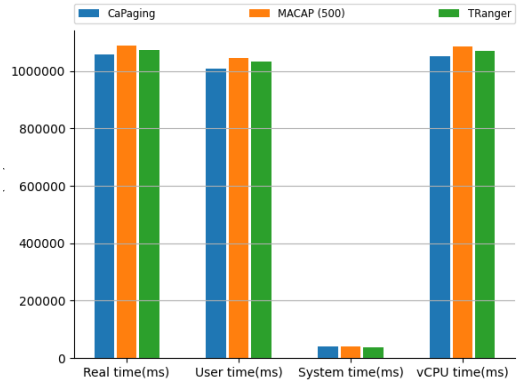


(δ') *Με 35% ήδη δεσμευμένη μνήμη.*

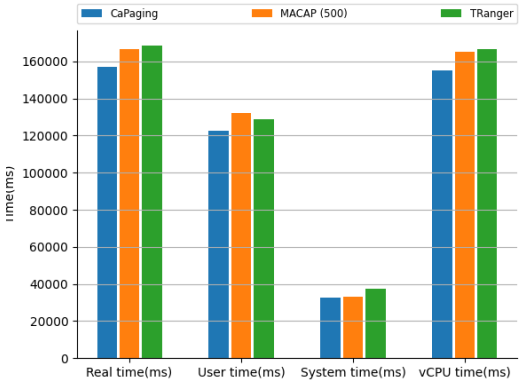
Σχήμα 5.11: *Χρόνοι εκτέλεσης για XSBench.*



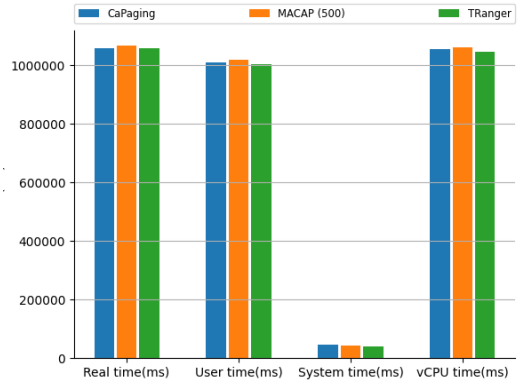
(α') *Με 50% ήδη δεσμευμένη μνήμη.*



(β') *Με 60% ήδη δεσμευμένη μνήμη.*

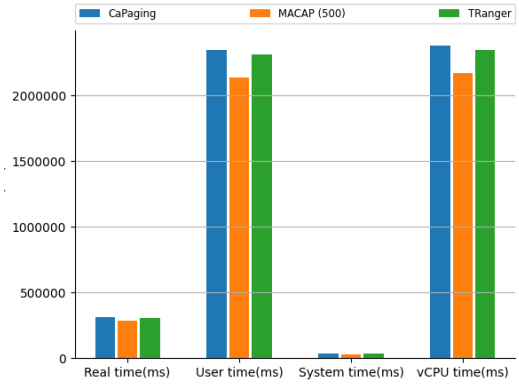


(γ') *Με 70% ήδη δεσμευμένη μνήμη.*

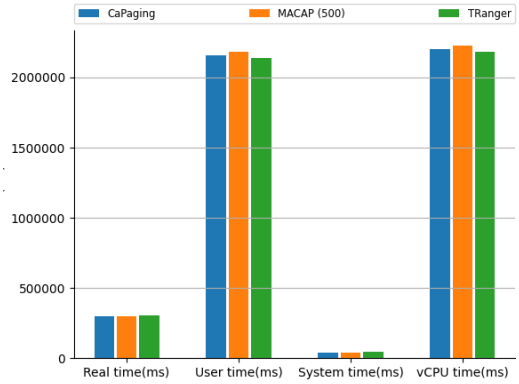


(δ') *Με 75% ήδη δεσμευμένη μνήμη.*

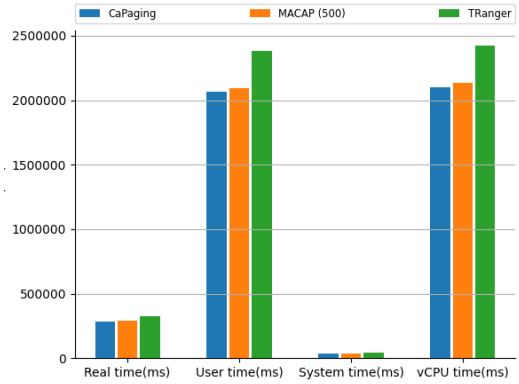
Σχήμα 5.12: *Χρόνοι εκτέλεσης για Liblinear.*



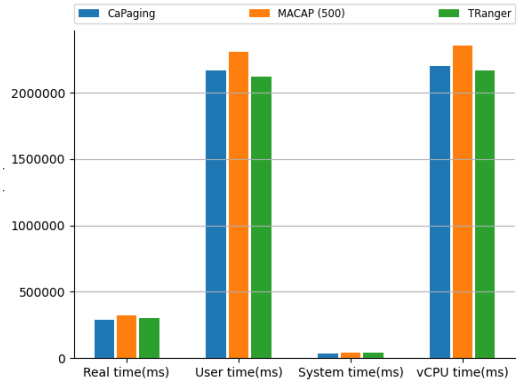
(α') *Με 10% ήδη δεσμευμένη μνήμη.*



(β') *Με 20% ήδη δεσμευμένη μνήμη.*



(γ') *Με 30% ήδη δεσμευμένη μνήμη.*



(δ') *Με 35% ήδη δεσμευμένη μνήμη.*

Σχήμα 5.13: *Χρόνοι εκτέλεσης για Hashjoin.*

5.4 Συμπεράσματα

Συμπερασματικά, από τα πειράματα που εκτελέσαμε, ο MACAP φαίνεται ότι επιτυγχάνει στον στόχο για τον οποίο σχεδιάστηκε, δηλαδή στην επίτευξη υψηλής μεταφραστικής συνέχειας ανεξάρτητα από την κατάσταση της μνήμης. Έχει άριστη επίδοση στην κάλυψη της μνήμης σε όλα τα σενάρια, και σχεδόν σε όλα καταφέρνει να έχει και την καλύτερη κάλυψη από τους τρεις μηχανισμούς. Βέβαια, και οι TRanger, CaPaging έχουν εξίσου καλές επιδόσεις, αλλά με περισσότερα ranges. Επομένως, ο MACAP εξασφαλίζει με μεγαλύτερη σιγουριά την καλή απόδοση μικρότερων range TLBs (π.χ. με λιγότερες από 64 καταχωρήσεις), το οποίο είναι πολύ σημαντικό για την μείωση της επιβάρυνσης που προσθέτει η αναζήτηση στο TLB Επιπλέον, είναι πιο βιώσιμο μακροπρόθεσμα, διότι υπάρχει μεγαλύτερο περιθώριο αύξησης του μεγέθους του TLB μελλοντικά, αν οι 32 ή 64 εγγραφές δεν είναι αρκετές για πιο πολύπλοκα προγράμματα όσο αυξάνονται και άλλο οι απαιτήσεις σε φυσική μνήμη.

Επιπλέον, ο MACAP φαίνεται ότι εκμεταλλεύεται σωστά τους δύο μηχανισμούς που τον απαρτίζουν και προσαρμόζεται κατάλληλα ανάλογα με το σενάριο εκτέλεσης. Όπου χρειάζεται, αφήνει αρκετές σελίδες να αποτύχουν ώστε να τις μετακινήσει μετά, δίχως συγχρόνως να καταλήγει να μετακινεί πλήθος σελίδων παρόμοιο με αυτό του TRanger. Ενώ σε περιπτώσεις μη αρκετά κατακερματισμένης μνήμης, αποφεύγει τις πολλές μετακινήσεις και βασίζεται κυρίως στα δυνατά σημεία που κληρονόμησε από τον CaPaging.

Αναλυτικότερη σύγκριση με CaPaging

Σε σχέση με τον Contiguity-aware Paging, ο μηχανισμός μας υπερισχύει σε κάθε περίπτωση όπου έχουμε έντονο κατακερματισμό μνήμης και υπάρχει αρκετή δεσμευμένη φυσική μνήμη. Αυτό τον κάνει πιθανώς καταλληλότερο από το CaPaging για χρήση σε περιπτώσεις όπου αναγκαστικά τρέχουν και άλλες διεργασίες παράλληλα και σε συστήματα που δεν υπάρχει μεγάλο περιθώριο συχνών επανεκκινήσεων,τα οποία πάσχουν από κατακερματισμό μνήμης λόγω των συχνών δεσμεύσεων-αποδεσμεύσεων μνήμης.

Επιπροσθέτως, ακόμα και σε περιπτώσεις όπου η φυσική μνήμη είναι αρκετά πιεσμένη και ο MACAP κάνει συχνή χρήση του μηχανισμού μετακίνησης σελίδων, βλέπουμε ότι συνεχίζει να πραγματοποιεί τις περισσότερες σωστές τοποθετήσεις μεγάλων σελίδων κατά την δέσμευση των σελίδων (Σχήματα 5.6-5.9). Σε αυτό πολύ σημαντική είναι η συνεισφορά της δυνατότητας προληπτικής δημιουργίας subVMAs, διότι προλαβαίνει πολλές λανθασμένες τοποθετήσεις κατά την αρχική δέσμευση των σελίδων.

Όμως, εξαιτίας των μετακινήσεων σελίδων, ο MACAP μπορεί να δημιουργήσει υπερβολικά πολλές αντιστοιχίσεις με 4K σελίδες, το οποίο αν και δεν επηρεάζει την απόδοση ενός range TLB, οδηγεί σε πολύ μεγαλύτερο πίνακα σελίδων. Ο μεγαλύτερος πίνακας σελίδων, λόγω του παραπάνω χώρου που χρειάζεται η αποθήκευσή του, αυξάνει το πλήθος των σελίδων του πυρήνα, οι οποίες δημιουργούν κατακερματισμό που μπορεί να δημιουργήσει πρόβλημα στην δημιουργία μεγάλων contiguous mappings.

Αναλυτικότερη σύγκριση με Translation Ranger

Σε σχέση με τον Translation Ranger, ο MACAP πετυχαίνει καλύτερη κάλυψη σχεδόν σε όλα τα σενάριο εκτέλεσης και όχι μόνο σε κατακερματισμένη μνήμη, όπως με το CaPaging. Επίσης, πετυχαίνει καλύτερη μεταφραστική συνέχεια με πολύ λιγότερες μετακινήσεις σελίδων. Αυτό το καταφέρνει, διότι μετακινεί μόνο τις λανθασμένα τοποθετημένες σελίδες.

Επίσης, ο MACAP λόγω ότι χρησιμοποιεί τις δυνατότητες του CaPaging, χτίζει σταδιακά τα contiguous mappings. Αντιθέτως, ο TRanger μπορεί σε κάποια φάση να αναδιοργανώσει σε νέα διαφορετικά ranges όλες τις φυσικές σελίδες της διεργασίας που έχουν ήδη δεσμευτεί. Για αυτό παρατηρήσαμε ότι ο TRanger σε όλα τα σενάρια εκτέλεσης, μετακινήσει πλήθος σελίδων όπου αθροιστικά το μέγεθός τους είναι πολύ μεγαλύτερο από το memory footprint του benchmark που εκτελούμε.

Κεφάλαιο 6

Μελλοντικές επεκτάσεις

Ολοκληρώνοντας την διπλωματική εργασία, θα παρουσιάσουμε μερικές προτάσεις και σενάρια για την επέκταση των δυνατοτήτων του MACAP:

- Μία από τις σημαντικότερες δυνατότητες του νέου μηχανισμού είναι η συντηρητική δημιουργία νέων subVMAs, αφήνοντας μεγάλες σελίδες να μην τοποθετηθούν στην κατάλληλη θέση μέχρι το πλήθος των ανεκτών αποτυχιών να φτάσει ένα κατώφλι. Στα πλαίσια της διπλωματικής εργασίας, περιοριστήκαμε να βρούμε μία τιμή (500) για το κατώφλι των ανεκτών αποτυχιών που να οδηγεί σε καλά αποτελέσματα και πορευτήκαμε στην αξιολόγηση του μηχανισμού αποκλειστικά με αυτήν την τιμή. Μία αξιόλογη μελλοντική εργασία στον MACAP είναι η βελτίωση του μηχανισμού ώστε να κάνει χρήση ενός δυναμικού κατωφλίου.

- Ο μηχανισμός που παρουσιάσαμε επικεντρώνεται στην επίτευξη μεταφραστικής συνέχειας σε σενάρια όπου προτιμάται η χρήση μεγάλων σελίδων. Όλα τα πειράματα μας έγιναν με τις κατάλληλες ρυθμίσεις ώστε να γίνεται, όπου είναι δυνατόν, η χρήση τους. Μία ενδιαφέρουσα μελέτη είναι η διερεύνηση της απόδοσης του μηχανισμού σε σενάρια όπου αποφεύγεται η χρήση μεγάλων σελίδων (π.χ. κάποιες βάσεις δεδομένων [6]) και η επέκταση του ώστε να διαχειρίζεται καλύτερα σελίδες βάσης.

- Η σχεδίαση μας βασίζεται αποκλειστικά στα subVMAs που δημιουργούνται κατά το allocations. Ενδιαφέρον θα είχε η υλοποίηση της δυνατότητας δημιουργίας subVMAs και από το νήμα πυρήνα, δηλαδή κατά την μετακίνηση των σελίδων. Η επέκταση αυτή θα δουλεύει βοηθητικά στο κύριο υπάρχον μηχανισμό δημιουργίας subVMAs.

- Ένα από τα ζητήματα για το οποίο δεν ασχοληθήκαμε με την πλήρης αντιμετώπισή του ήταν οι μη-μετακινήσιμες σελίδες. Επομένως, για την περαιτέρω βελτίωση της μεταφραστικής συνέχειας ανεξάρτητα από το σενάριο εκτέλεσης, είναι χρήσιμη η διεξαγωγή έρευνας με σκοπό έναν αποδοτικό σχέδιο για τον περιορισμό τους σε ένα μικρό συγκεκριμένο τμήμα της μνήμης.

- Ο μηχανισμός, ως έχει, δεν μπορεί να είναι ενεργοποιημένος συγχρόνως με διάφορους μηχανισμούς του λ.σ. Linux, όπως ο khugepaged και ο kcompactd[11]. Θα είχε ενδιαφέρον η διερεύνηση μίας συνολικότερης ενσωμάτωσης του μηχανισμού σε όλες τις πτυχές του μηχανισμού διαχείρισης μνήμης του Linux.

- Αξιολόγηση του μηχανισμού σε NUMA αρχιτεκτονικές πολλών κόμβων με κατάλληλη επέκταση της ανταλλαγής σελίδων, διότι ο κλασσικός TRanger δεν είναι αποδοτικός σε NUMA αρχιτεκτονικές (πολλών κόμβων μνήμης)[4].
- Βελτιστοποίηση των δομών δεδομένων του μηχανισμού και της πρόσβασής του σε αυτές. Για παράδειγμα, χρήση δέντρων κατάλληλου τύπου για την αποθήκευση των subVMAs και αντί για ένα κοινό lock ανά VMA, να έχουμε μία πιο fine-grained προσέγγιση.

Κεφάλαιο 7

Σχετική ερευνητική βιβλιογραφία

Στην διπλωματική εργασία περιοριστήκαμε στην παρουσίαση των μηχανισμών RMM, CaPaging και TRanger, των οποίων η ανάλυση είναι απαραίτητη για την κατανόηση της εργασία μας και ειδικά της πρότασή της, τον ΜΑΆΠ. Πέρα όμως από αυτούς τους μηχανισμούς, για το πρόβλημα της περιορισμένης κάλυψης της μνήμης από το TLB και την δημιουργία συνεχόμενων αντιστοιχίσεων, έχουν υπάρξει αρκετές άλλες αξιόλογες προτάσεις σε επίπεδο λογισμικού και υλικού. Η πρόταση των **Direct Segments**[12] αφορά μία συνολική υλοποίηση σε επίπεδο υλικού και λογισμικού για την δυνατότητα δημιουργίας μίας αντιστοίχισης για ένα μεγάλος εύρος του χώρου εικονικού διευθύνσεων, ενώ το υπόλοιπο θα χρησιμοποιεί σελίδες κλασσικά. Μία άλλη πρόταση είναι το **Hybrid TLB Coalescing**[13], το οποίο αποτελεί μία νέα τεχνική, σε επίπεδο λογισμικού και υλικού, συγκέντρωσης σελίδων σε contiguous mappings. Τέλος, υπάρχει και ο μηχανισμός **SpOT**[3], που πρόκειται για έναν απλό μηχανισμό υλικού που εκμεταλλεύεται την μεταφραστική συνέχεια που δημιουργούν μηχανισμοί όπως οι TRanger και CaPaging, με σκοπό να προβλέπει χαμένες μεταφράσεις διευθύνσεων.

Βιβλιογραφία

[1] Abhishek Bhattacharjee και Daniel Lustig. *Architectural and Operating System Support for Virtual Memory*. *Synthesis Lectures on Computer Architecture*, 12(5):1–175, 2017.

[2] Vasileios Karakostas, Jayneel Gandhi, Furkan Ayar, Adrián Cristal, Mark D. Hill, Kathryn S. McKinley, Mario Nemirovsky, Michael M. Swift και Osman Ünsal. *Redundant Memory Mappings for Fast Access to Large Memories*. *SIGARCH Comput. Archit. News*, 43(3S):66–78, 2015.

[3] Chloe Alverti, Stratos Psomadakis, Vasileios Karakostas, Jayneel Gandhi, Konstantinos Nikas, Georgios Goumas και Nectarios Koziris. *Enhancing and Exploiting Contiguity for Fast Memory Virtualization*. *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, σελίδες 515–528, 2020.

[4] Zi Yan, Daniel Lustig, David Nellans και Abhishek Bhattacharjee. *Translation Ranger: Operating System Support for Contiguity-Aware TLBs*. *Proceedings of the 46th International Symposium on Computer Architecture*, σελίδες 698–710, New York, NY, USA, 2019. Association for Computing Machinery.

[5] *Huge pages part 1 (Introduction)*. https://lwn.net/Articles/374424/. Ημερομηνία πρόσβασης: 16-09-2021.

[6] *Settling the Myth of Transparent HugePages for Databases*. https://www.percona.com/blog/2019/03/06/settling-the-myth-of-transparent-hugepages-for-databases/. Ημερομηνία πρόσβασης: 16-09-2021.

[7] Daniel P. Bovet και Marco Cesati. *Understanding the Linux Kernel*. O'Reilly Media, Inc, Sebastopol, California, 3η έκδοση, 2006.

[8] *Transparent hugepages*. https://lwn.net/Articles/359158/. Ημερομηνία πρόσβασης: 18-9-2021.

[9] Rong En Fan, Kai Wei Chang, Cho Jui Hsieh, Xiang Rui Wang και Chih Jen Lin. *LIBLINEAR: A library for large linear classification*. *the Journal of machine Learning research*, 9:1871–1874, 2008.

[10] John R Tramm, Andrew R Siegel, Tanzima Islam και Martin Schulz. *XSBench-the development and verification of a performance abstraction for Monte Carlo reactor analysis*. *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)*, 2014.

[11] *Memory compaction*. https://lwn.net/Articles/368869/. Ημερομηνία πρόσβασης: 20-9-2021.

[12] Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill και Michael M. Swift. *Efficient Virtual Memory for Big Memory Servers*. *SIGARCH Comput. Archit. News*, 41(3):237–248, 2013.

[13] Chang Hyun Park, Taekyung Heo, Jungi Jeong και Jaehyuk Huh. *Hybrid TLB coalescing: Improving TLB translation coverage under diverse fragmented memory allocations*. *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, σελίδες 444–456, 2017.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

VPN	Virtual Page Number
PFN	Page Frame Number
VMA	Virtual Memory Area
PTE	Page Table Entry
HPC	Hyper Perfomance Computing
CaP	Contiguity-aware Paging
TR	Translation Ranger
MACAP	Migration-assisted Contiguity-aware Paging
λ.σ.	λειτουργικό σύστημα
π.χ.	παραδείγματος χάριν
κ.λ.π.	και λοιπά

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος
μεταφραστική συνέχεια	translation contiguity
εύρος συνεχόμενων αντιστοιχίσεων	contiguous mapping
αποτύπωμα μνήμης	memory footprint
συμπυκνωμένη περιοχή	coalesced region
κρυφή μνήμη	cache