ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Μελέτη Επίδοσης και Ευέλικτες Πολιτικές Τοποθέτησης Δεδομένων σε NUMA Αρχιτεκτονικές μη Πτητικών Μνημών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Τόφαλος Φίλιππος

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

Αθήνα, Ιούλιος 2023

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Μελέτη Επίδοσης και Ευέλικτες Πολιτικές Τοποθέτησης Δεδομένων σε NUMA Αρχιτεκτονικές μη Πτητικών Μνημών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Τόφαλος Φίλιππος

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Ιουλίου 2023.

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

Αθήνα, Ιούλιος 2023

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

Περίληψη

Η εμπορική διαθεσιμότητα των μη πτητικών μνημών (Persistent Memory ή PMem) έφερε στην αγορά των σύγχρονων υπολογιστικών συστημάτων συσκευές οι οποίες συνδυάζουν μεγάλη χωρητικότητα, συγκράτηση δεδομένων σε συνθήκες έλλειψης τροφοδοσίας, και επιδόσεις μόνο μια τάξη μεγέθους μακριά από την τεχνολογία DRAM. Τα συγκεκριμένα χαρακτηριστικά επιτρέπουν την αξιοποίηση τους τόσο ως μνήμη όσο και ως μέσο αποθήκευσης, και προορίζονται για χρήση κυρίως σε συστήματα αρχιτεκτονικής NUMA. Όμως, το κόστος απομακρυσμένης προσπέλασης σε μια NUMA διάταξη μη πτητικών μνημών είναι πολύ μεγαλύτερο σε σύγκριση με μια ανομοιόμορφη DRAM προσπέλαση, και παρουσιάζει μοναδικές ιδιορρυθμίες, ειδικά για προσπελάσεις εγγραφών.

Στη διπλωματική αυτή μελετάμε τις προαναφερθείσες διατάξεις, όπου η μη πτητική μνήμη χρησιμοποιείται ως μέσο αποθήκευσης άμεσης προσπέλασης (Direct Access – DAX). Εξετάζουμε το ρόλο του συστήματος αρχείων ext4 στην τοποθέτηση δεδομένων στους διάφορους NUMA κόμβους, και διαπιστώνουμε ότι η έλλειψη επίγνωσης του για την υποκείμενη NUMA τοπολογία των συσκευών είναι σε θέση να επιβαρύνει σημαντικά την επίδοση τους. Έτσι, επεκτείνουμε τους αλγόριθμους που διαθέτει για δέσμευση block (μετα)δεδομένων ώστε να έχουν χαρακτηριστικά "NUMA awareness" και να γίνονται οι δεσμεύσεις όσο το δυνατό στις τοπικές μη πτητικές συσκευές του κόμβου κάθε νήματος. Η επέκταση αυτή προσφέρει πάνω από δύο φορές βελτίωση επίδοσης για εφαρμογές μεγάλης έντασης I/O.

Τέλος, επεκτείνουμε την λογική συλλογής στατιστικών δεδομένων ανά διεργασία για τις προσβάσεις στο storage layer ώστε να υποστηρίζουν την ανά κόμβο καταγραφή τους σε λειτουργία DAX. Βάσει των στατιστικών αυτών, σκιαγραφούμε την ανάπτυξη μιας userspace βιβλιοθήκης που θα τοποθετεί δυναμικά τα νήματα μιας εφαρμογής στους κατάλληλους κόμβους, για την βελτιστοποίηση της αξιοποίησης του συνολικού bandwidth των συσκευών.

Λέξεις κλειδιά

NUMA, μη πτητικές μνήμες, συστήματα αρχείων, πυρήνας Linux

Abstract

The commercial availability of non-volatile memory DIMMs (Persistent Memory or Pmem) has provided the landscape of modern computing with devices that combine large capacity, data persistence and performance on the order of DRAM. These characteristics enable utilizing these devices as either memory or storage, and they are intended to be used in NUMA systems. Regardless, the cost of remote access of persistent memory devices arranged in a NUMA topology is much higher compared to non uniform access of DRAM, and said cost presents certain peculiarities, specifically for write accesses.

In this diploma thesis we study the aforementioned arrangement of persistent memory devices in a NUMA setting, in which the devices are used as direct access (DAX) enabled storage. We examine the role of the ext4 filesystem in placing data among NUMA nodes, realizing that its lack of awareness regrading the underlying NUMA topology can noticeably impact the performance of the devices. Given this, we extend the (meta)data allocation algorithms present in ext4 to be NUMA aware and allow for allocating blocks at devices in the same NUMA node as the writing thread. This extension offers up to two times performance gain in I/O intensive workloads.

Finally, we extend the per-process I/O statistics collection for accesses to the storage layer so as to additionally support collecting this information per NUMA node and for DAX mode as well. Based on these statistics, we outline the development of a userspace library that dynamically places application threads to the suitable NUMA nodes for the purpose of better utilizing the aggregate bandwidth of the persistent memory devices.

Keywords

NUMA, persistent memory, filesystems, Linux kenrel

Περιεχόμενα

Κατάλογος Πινάκων

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός Εργασίας

Η τυπική δομή ενός σύγχρονου υπολογιστικού συστήματος περιλαμβάνει μεταξύ άλλων μια μονάδα επεξεργασίας (CPU), μη πτητικά μέσα αποθήκευσης για συγκράτηση δεδομένων μεταξύ κύκλων λειτουργίας του συστήματος, και (πτητικές) μνήμες RAM που συνδέονται μέσω ενός διαύλου επικοινωνίας με τον επεξεργαστή.

Εμφανίζονται δύο βασικά προβλήματα σε αυτήν την δομή: Το πρώτο πρόβλημα είναι ότι υπάρχει χάσμα μεταξύ της επίδοσης των επεξεργαστών και των μνημών, το οποίο μόνο εντείνεται από την αύξηση του αριθμού και της ταχύτητας των επεξεργαστικών πυρήνων. Το δεύτερο είναι ότι υπάρχει επιπλέον σημαντικό χάσμα επίδοσης μεταξύ των μη πτητικών μέσων αποθήκευσης και των μνημών.

Το πρώτο πρόβλημα αντιμετωπίζεται σε επίπεδο αρχιτεκτονικής των υπολογιστικών συστημάτων, με την θεώρηση της αρχιτεκτονικής NUMA. Προηγουμένως, όλοι οι επεξεργαστικοί πυρήνες πραγματοποιούσαν προσβάσεις στο ίδιο σύνολο μνημών, εξαντλώντας γρήγορα την επίδοση τους. Στην αρχιτεκτονική NUMA, οι πυρήνες πλέον χωρίζονται σε ομάδες, τους λεγόμενους NUMA κόμβους, με την κάθε μια να διαθέτει δικό της (τοπικό) σύνολο μνημών και μέσων αποθήκευσης. Για διασφάλιση συνοχής των δεδομένων του συστήματος, τοποθετείται ένα δίκτυο διασύνδεσης μεταξύ των NUMA κόμβων. Αυτό οδηγεί σε ανομοιόμορφες συνθήκες πρόσβασης αναλόγως με το αν αυτή είναι στο τοπικό σύνολο μνημών του αντίστοιχου επεξεργαστικού πυρήνα, ή αν χρειάζεται δεδομένα ενός άλλου κόμβου, άρα απαιτείται μεσολάβηση του δικτύου διασύνδεσης. Λέμε ότι έχουμε τοπικές και απομακρυσμένες προσβάσεις αντίστοιχα.

Το δεύτερο πρόβλημα μπορεί να λυθεί σε επίπεδο της τεχνολογίας του μέσου αποθήκευσης, και είναι υπό διαρκή διερεύνηση της ερευνητικής κοινότη-

τας. Λίγα χρόνια πριν, έγιναν εμπορικά διαθέσιμες οι πρώτες μη πτητικές μνήμες, συγκεκριμένα τα Intel Optane DIMM, ένα προϊόν που υπόσχεται το χαρακτηριστικό της μη πτητικότητας με επίδοση πιο κοντινή σε αυτήν της τεχνολογίας DRAM παρά σε προϋπάρχουσες τεχνολογίες μέσων αποθήκευσης.

Τόσο η αρχιτεκτονική NUMA όσο και οι η τεχνολογία των μη πτητικών μνημών απευθύνονται σε υπολογιστικά συστήματα υψηλής επίδοσης, τα οποία χρειάζονται αμφότερα την αποδοτικότερη αξιοποίηση του συστήματος μνήμης που παρέχει η πρώτη, αλλά και την ταχύτερη προσπέλαση αποθηκευμένων δεδομένων που προσφέρει η δεύτερη. Παρουσιάζεται λοιπόν η ανάγκη να μελετηθεί ο τρόπος με τον οποίον αλληλεπιδρούν οι δύο προαναφερθείς παράγοντες, ώστε να προσαρμόσουμε κατάλληλα την υπάρχουσα υποδομή λογισμικού με τρόπο που θα αποτρέπει την αρνητική συνεργεία τους.

Συγκεκριμένα, ολικός σκοπός είναι ο τελικός χρήστης ενός NUMA συστήματος να μπορεί να αξιοποιήσει το σύνολο των πόρων του συστήματος με όση περισσότερη διαφάνεια της υποκείμενης αρχιτεκτονικής. Επομένως, για να χρησιμοποιηθεί το σύνολο των μη πτητικών μνημών αποδοτικά δεδομένου μη ομοιόμορφων προσβάσεων σε αυτές, χρειάζεται να εντοπίσουμε τις σχετικές ιδιοτροπίες και παθογένειες που μπορούν να προκύψουν. Τελικός σκοπός είναι να ενσωματώσουμε κάποια λογική διαχείρισης ή αποφυγής των ιδιαζουσών περιπτώσεων στην υλοποίηση των κατάλληλων μηχανισμών αφαίρεσης που παρέχει το λειτουργικό για πρόσβαση στις μη πτητικές μνήμες. Εν προκειμένω ενδιαφερόμαστε για την λειτουργία των μη πτητικών μνημών σαν μέσο αποθήκευσης, και για αυτό ο μηχανισμός τον οποίον στοχεύουμε να αναδιαμορφώσουμε είναι το σύστημα αρχείων.

1.2 Συνεισφορά

Σε αυτήν τη διπλωματική εργασία επεκτείναμε κατάλληλα το σύστημα αρχείων ext4 προκειμένου να εξετάσουμε και να αναδείξουμε την αποτελεσματικότητα της δέσμευσης χώρου με επίγνωση της κατανομής των μη πτητικών μνημών στους επιμέρους NUMA κόμβους.

Μελετήσαμε την συμπεριφορά των μη πτητικών συσκευών Intel Optane DIMM προσδιορίζοντας μια θεμελιώδη παθογένεια όταν αυτές δέχονται αλληλουχίες ετερογενών απομακρυσμένων προσβάσεων. Δεδομένου αυτής, ανασχεδιάσαμε κατάλληλα τις ρουτίνες δέσμευσης (μετα)δεδομένων του ext4 ώστε να επιφέρουν αποδοτικότερες προσβάσεις όταν το σύστημα αρχείων έχει σχηματιστεί επί κατάλληλα διαμορφωμένης συσκευής που ενώνει με γραμμικό τρόπο όλες τις μη πτητικές συσκευές του συστήματος. Η συσκευή αυτή προκύπτει μέσω του οδηγού device mapper του πυρήνα Linux.

Αναπτύχθηκε μια πλήρη μεθοδολογία αξιολόγησης της ορθότητας και α-

ποδοτικότητας των επεκτάσεων που πραγματοποιήθηκαν, αναδεικνύοντας ότι η τροποποιημένη εκδοχή του ext4 είναι ικανή να επιφέρει σε ορισμένα σενάρια εκτέλεσης διεργασιών I/O έως και δύο φορές καλύτερη αξιοποίηση του αθροιστικού bandwidth των μη πτητικών συσκευών σε σχέση με την αρχική του εκδοχή και για διαφορετικούς τρόπους συνένωσης των συσκευών.

Επιλέξαμε το σύστημα αρχείων ext4 καθώς είναι ευρέως διαδεδομένο και προσφέρει την δυνατότητα επιλογής αξιοποίησης ή μη της page cache σε επίπεδο μεμονωμένων αρχείων (χαρακτηριστικό per-file DAX). Προσδίδοντας στο ext4 χαρακτηριστικά NUMA awareness, κατασκευάζουμε υποδομή για μελλοντική μελέτη του τρόπου υπό συνθηκών αξιοποίησης της page cache προς άρση των επιβλαβών άμεσων απομακρυσμένων προσβάσεων στις μη πτητικές μνήμες.

Προκειμένου να γίνει πληρέστερη η υποδομή στήριξης μελλοντικής έρευνας στο θέμα που εξετάζει η παρούσα διπλωματική, επεκτάθηκε το σύστημα συλλογής στατιστικών I/O του πυρήνα Linux ώστε σε συνεργασία με τον device mapper να υποστηρίζει την συλλογή των στατιστικών σε επίπεδο NUMA κόμβων και να είναι συμβατή με την λειτουργία DAX.

Βάσει της επέκτασης του συστήματος συλλογής στατιστικών IO, διαμορφώθηκε η βάση ενός μηχανισμού στο userspace για συλλογή δεδομένων εκτέλεσης επιλεγμένων νημάτων και δυναμικής τοποθέτησης τους στους κατάλληλους NUMA κόμβους. Παρόλο που ο πηγαίος κώδικας για τον μηχανισμό αυτό είναι προς το παρόν απλοϊκός, στην εργασία αυτή αναλύεται ο τρόπος επέκτασης του και ποιες είναι οι σχετικές προκλήσεις που πρέπει να επιλυθούν για να εξελιχθεί στον απαραίτητο βαθμό.

Τέλος, εκτός των ήδη αναφερθέντων πιθανών επεκτάσεων της εργασίας, ονομαστικά την ολοκλήρωση του userspace μηχανισμού και την μερική αξιοποίηση της page cache για εξομάλυνση των παθογενειών στην μη ομοιόμορφη πρόσβαση, αναφέρεται συνοπτικά και ο τρόπος με τον οποίο θα μπορούσαμε να προσδώσουμε χαρακτηριστικά NUMA τοπικότητας ακόμα και στις εγγραφές ήδη δεσμευμένων block δεδομένων.

1.3 Οργάνωση Κειμένου

Κεφάλαιο 2: Θεωρητικό Υπόβαθρο

Στο κεφάλαιο αυτό βλέπουμε πιο αναλυτικά την έννοια της αρχιτεκτονικής NUMA, τις διάφορες τεχνολογίες μνήμης και πως συγκρίνονται με αυτές οι μη πτητικές μνήμες, στοιχεία αρχιτεκτονικής των μνημών αυτών, και λεπτομέρειες υλοποίησης του ext4, η κατανόηση των οποίων συμβάλει στην επέκταση του συστήματος αρχείων.

Κεφάλαιο 3: Μεθοδολογία

Στο κεφαλαίο αυτό παρουσιάζονται όλα τα διαθέσιμα εργαλεία και οι μέθοδοι που αξιοποιήθηκαν για την ανάπτυξη του πυρήνα Linux και την διεξαγωγή των πειραμάτων. Συγκεκριμένα, θα περιγραφεί συνοπτικά η λειτουργία των εργαλείων FIO και Filebench, το αυτοματοποιημένο σύστημα εκτέλεσης πειραμάτων του πηγαίου κώδικα της εργασίας, καθώς και η διαμόρφωση υποδομής ανάπτυξης του πυρήνα και οι μέθοδοι ανάλυσης της συμπεριφοράς του.

Κεφάλαιο 4: Πειράματα

Στο κεφάλαιο αυτό αναλύονται τα πειράματα που αναδεικνύουν τις παθογένειες στις ανομοιόμορφες προσβάσεις των μη πτητικών μνημών, οι οποίες αποτέλεσαν μέρος της αφόρμησης για την παρούσα εργασία.

Κεφάλαιο 5: Κίνητρο της Εργασίας

Στο κεφάλαιο αυτό παρουσιάζονται σε αφαιρετικό επίπεδο οι στόχοι που θέτουμε για την παρούσα εργασία βάσει των αποτελεσμάτων του κεφαλαίου 4.

Κεφάλαιο 6: Υλοποίηση

Στο κεφάλαιο αυτό περιγράφονται οι λεπτομέρειες σχεδιασμού και υλοποίησης της υποδομής λογισμικού αυτής της διπλωματικής εργασίας, και καταγράφονται οι πιθανές επεκτάσεις του μηχανισμού στο userspace.

Κεφάλαιο 7: Αξιολόγηση

Στο κεφάλαιο αυτό περιγράφουμε όλη την διαδικασία αξιολόγησης των αλλαγών που έγιναν στο σύστημα αρχείων ext4, τόσο σε επίπεδο ορθότητας της υλοποίησης (ευστάθεια εκτέλεσης και σεβασμός των επιθυμητών ιδιοτήτων κατά την λειτουργία του) όσο και σε επίπεδο επίδοσης, αξιοποιώντας τα εργαλεία του κεφαλαίου 3.

Κεφάλαιο 8: Σχετική Έρευνα

Στο κεφάλαιο αυτό παρουσιάζεται η σχετική έρευνα πάνω στις μη πτητικές μνήμες. Συγκεκριμένα, θα παρουσιαστούν τα συστήματα αρχείων που έχουν αναπτυχθεί ειδικά για αυτές, και κάποιες δουλειές που στοχεύουν συγκεκριμένα στην επίδοση των μη πτητικών μνημών σε σχέση με την αρχιτεκτονική NUMA.

Κεφάλαιο 9: Επίλογος

Στο τελευταίο κεφάλαιο συνοψίζουμε την δουλειά που έγινε στην διπλωματική αυτή και αναφερόμαστε στις πιθανές επεκτάσεις της.

1.4 Πηγαίος Κώδικας

Ο πηγαίος κώδικας της εργασίας μπορεί να βρεθεί στο ακόλουθο repository:

`https://github.com/PhTof/Diploma`

Συχνά στο παρόν έγγραφο θα γίνεται αναφορά σε αρχεία του repository αυτού. Σε ορισμένες περιπτώσεις μπορεί να γίνεται αναφορά και στον αρχικό κώδικα της έκδοσης 5.13 του πυρήνα Linux αντί για την τροποποιημένη εκδοχή που βρίσκεται μεταξύ άλλων στον πηγαίο κώδικα της εργασίας. Θα σημειώνεται ρητά ποια από τις δύο περιπτώσεις εννοούμε.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

Σε αυτό το κεφάλαιο θα παρουσιάσουμε το βασικό θεωρητικό υπόβαθρο που χρειάζεται για την καλύτερη κατανόηση της παρούσας εργασίας και της υπολογιστικής υποδομής που αξιοποιεί.

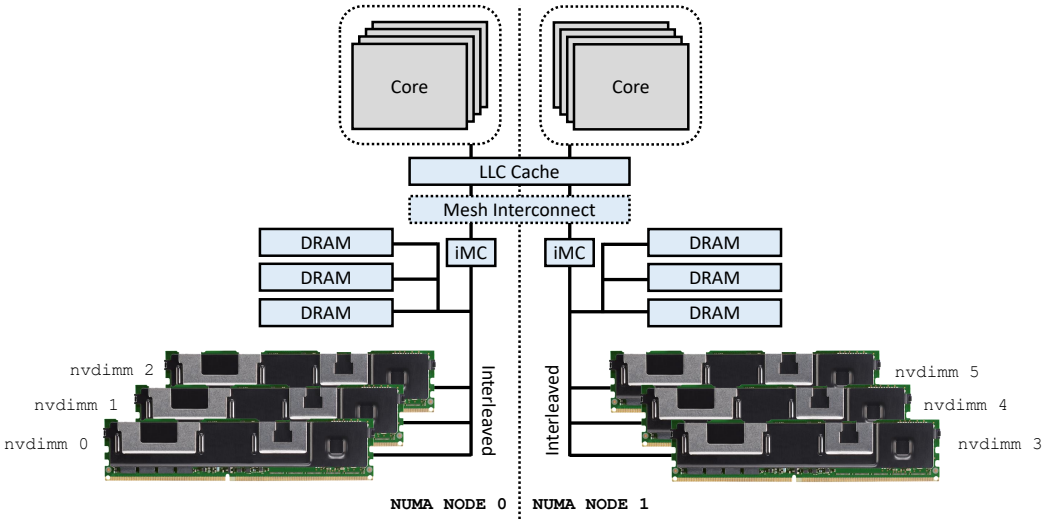
2.1 Αρχιτεκτονική NUMA

Από πολύ νωρίς στην ιστορία διαμόρφωσης των υπολογιστικών συστημάτων άρχισε να σημειώνεται απόκλιση μεταξύ του ρυθμού λειτουργίας των επεξεργαστών και της ικανότητας των μνημών να διεκπεραιώνουν αιτήματα πρόσβασης. Η άμεση λύση για την συντήρηση του ρυθμού κλιμακωσιμότητας των υπολογιστικών συστημάτων υπήρξε η διαμεσολάβηση μικρότερων, γρήγορων μονάδων προσωρινής αποθήκευσης (cache) μεταξύ επεξεργαστή και μνημών [20].

Ωστόσο, ακόμα και η εξέλιξη των προσωρινών αυτών μνημών δεν μπόρεσε να καλύψει πλήρως τις ανάγκες που προέκυψαν από την ανάπτυξη πολυεπεξεργαστικών συστημάτων, που οδήγησε σε όλο και αυξανόμενους σε πλήθος επεξεργαστικούς πυρήνες να ανταγωνίζονται για προσβάσεις στην μνήμη οδηγώντας σε εκφυλισμό της επίδοσης της.

Μια από τις λύσεις σε αυτό το πρόβλημα υπήρξε η ανάπτυξη συστημάτων όπου συγκεκριμένα υποσύνολα επεξεργαστικών πυρήνων, που αντιστοιχούν σε έναν NUMA κόμβο, διαθέτουν δική τους, τοπική μνήμη στην οποία έχουν άμεση πρόσβαση. Οι επεξεργαστές διαφορετικών NUMA κόμβων επικοινωνούν μεταξύ τους μέσω ενός δικτύου διασύνδεσης, δια του οποίου μεταφέρονται δεδομένα σε περίπτωση ανάγκης πρόσβασης ενός επεξεργαστή στην μνήμη ενός άλλου NUMA κόμβου.

Όπως είναι φυσικό, η ταχύτητα εξυπηρέτησης ενός αιτήματος πρόσβασης στην μνήμη εξαρτάται από το αν ο NUMA κόμβος στον οποίο είναι προσαρτη-



Σχήμα 2.1: Μη πτητικές μνήμες σε αρχιτεκτονική NUMA (διαμόρφωση AppDirect)

μένη η μονάδα μνήμης είναι ο τοπικός του αιτούμενου επεξεργαστή ή όχι. Επομένως, η αρχιτεκτονική ονομάζεται μη ομοιόμορφης προσπέλασης στην μνήμη (Non-Uniform Memory Access).

Στο σχήμα 2.1 φαίνεται η σχηματική αναπαράσταση ενός συστήματος NUMA με 2 κόμβους, με εμφανή τον τρόπο διαχωρισμού μεταξύ των διαφορετικών κόμβων σε επίπεδο αρχιτεκτονικής του συστήματος.

2.2 Τεχνολογίες Μέσων Αποθήκευσης

Για ευκολία και πληρότητα αναφοράς στην συνέχεια, παραθέτουμε και εξηγούμε κάποιες από τις μετρικές που χαρακτηρίζουν ένα μέσο αποθήκευσης ή μια μνήμη:

- **Ρυθμαπόδοση** (αλλιώς διαμεταγωγή ή bandwidth): Ο ρυθμός με τον οποίο διαβάζουμε ή γράφουμε δεδομένα στο εκάστοτε μέσο.
- **Λανθάνων χρόνος** (αλλιώς χρόνος αδράνειας ή latency): Το χρονικό διάστημα που μεσολαβεί από ένα αίτημα εγγραφής ή ανάγνωσης στο μέσο μέχρι την διεκπεραίωση του, δηλαδή την γνωστοποίηση ολοκλήρωσης της εγγραφής ή την μεταφορά των ζητούμενων δεδομένων αντιστοίχως.
- **Πτητικότητα δεδομένων**: Με τον όρο πτητικότητα αναφερόμαστε στην ιδιότητα αδυναμίας συγκράτησης των αποθηκευμένων δεδομένων

σε συνθήκες έλλειψης τροφοδοσίας. Η έννοια της μη πτητικότητας απορρέει με λογικό τρόπο από τον προηγούμενο ορισμό. Πτητικές (volatile) συσκευές αποτελούν οι μονάδες DRAM, και μη πτητικές οι σκληροί δίσκοι ή μέσα αρχιτεκτονικής NAND flash.

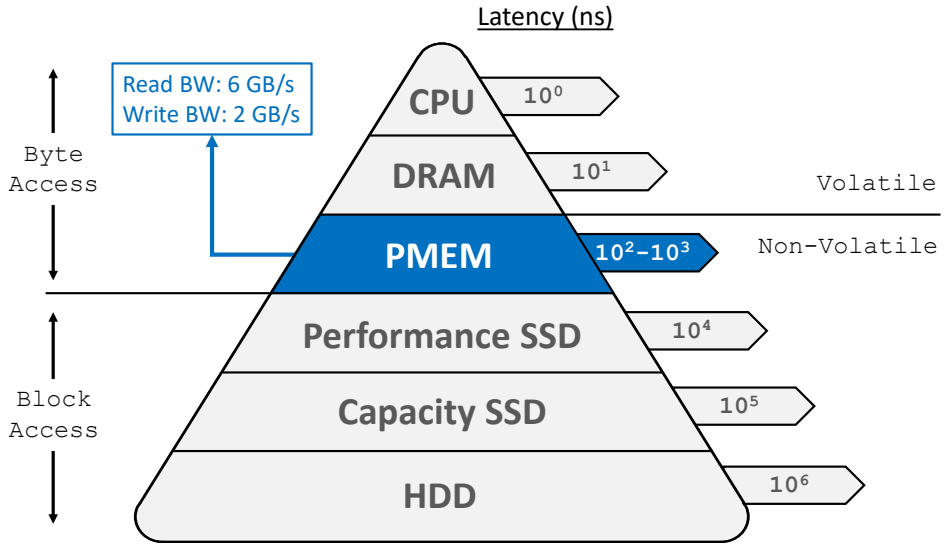
- **Πυκνότητα**: Η πληροφορία που μπορεί να αποθηκευτεί επί μιας μονάδας φυσικού χώρου της συσκευής αποθήκευσης, μετρημένη σε δυαδικά ψηφία (bit). Η μονάδα αυτή μπορεί να είναι το συνολικό εμβαδό των ολοκληρωμένων κυκλωμάτων της συσκευής.

Επιπροσθέτως, διαθέτουμε και ένα λειτουργικό χαρακτηριστικό, το οποίο αφορά περισσότερο τον τρόπο πρόσβασης δεδομένων επί της συσκευής, το λεγόμενο data access granularity. Αυτό αναλογεί στον ελάχιστο όγκο πληροφορίας που μπορεί να μεταφερθεί μέσω μιας πρόσβασης. Θα μας απασχολήσουν οι όροι byte granularity και block granularity. Με τον πρώτο εννοούμε ότι μπορούμε να έχουμε προσβάσεις σε επίπεδο byte, ή ισοδύναμα σε ένα οποιοδήποτε εύρος από byte. Με τον δεύτερο εννοούμε ότι όλες οι προσβάσεις γίνονται με μονάδα μεταφοράς δεδομένων ένα block συγκεκριμένου μεγέθους.

Θα δούμε ότι η έννοια του granularity εμπεριέχεται στον σχεδιασμό συστημάτων αρχείων με την έννοια του Direct Access (DAX) για συσκευές που υποστηρίζουν byte addressability.

Πριν τις μη πτητικές μνήμες διακρίνουμε σαφή διαχωρισμό μεταξύ των εμπορικά διαθέσιμων συσκευών αποθήκευσης δεδομένων και μνημών. Από την μια μεριά έχουμε τις μνήμες τυχαίας προσπέλασης (DRAM) που διαθέτουν byte granularity, υψηλή ρυθμαπόδοση, χαμηλό λανθάνοντα χρόνο αλλά και χαμηλή πυκνότητα. Από την άλλη μεριά, έχουμε συσκευές αποθήκευσης όπως οι σκληροί δίσκοι ή οι SSD διαφόρων ειδών, που σε αντίθεση με την DRAM διαθέτουν block granularity, κατά τάξεις μεγέθους χαμηλότερη ρυθμαπόδοση και μεγαλύτερο λανθάνοντα χρόνο, αλλά έχουν αυξημένη χωρητικότητα και είναι μη πτητικά μέσα.

Μεταξύ μνημών (όπως θα αναφέρουμε άτυπα την DRAM) και παραδοσιακών μέσων αποθήκευσης υπάρχει ένα χάσμα επίδοσης αν λάβουμε ως μετρικές το bandwidth και το latency. Αυτό το χάσμα προσπάθησαν να καλύψουν οι μη πτητικές μνήμες, συνδυάζοντας τα επιθυμητά χαρακτηριστικά των επιμέρους ειδών συσκευών. Συγκεκριμένα, διαθέτουν byte granularity, την ιδιότητα της μη πτητικότητας, και χαρακτηριστικά επίδοσης που πλησιάζουν σε τάξη μεγέθους αυτά της DRAM. Το γεγονός αυτό τους προσδίδει εφαρμογές τόσο ως μέρος της ιεραρχίας προσωρινής αποθήκευσης δεδομένων (με την DRAM να λειτουργεί ως direct-mapped cache των μη πτητικών μνημών, σε έναν τρόπο λειτουργίας που αναφέρεται ως memory mode) όσο και μέσου αποθήκευσης δεδομένων (το οποίο θα αναφέρουμε ως AppDirect mode).



Σχήμα 2.2: Η τοποθέτηση των μη πτητικών μνημών στην ιεραρχία τεχνολογιών μέσων αποθήκευσης

2.3 Χαρακτηριστικά των μη πτητικών μνημών

Στην προηγούμενη ενότητα είδαμε πιο αφαιρετικά την τοποθέτηση των μη πτητικών μνημών στην ιεραρχία των διάφορων τεχνολογιών αποθήκευσης δεδομένων (πτητικών και μη). Εδώ θα επιχειρήσουμε να παρουσιάσουμε πιο αναλυτικά και κάποια στοιχεία αρχιτεκτονικής των μνημών, μεταξύ άλλων. Αν και δεν αξιοποιούνται άμεσα στην υλοποίηση της εργασίας όλα όσα θα αναφερθούν αμέσως μετά, προσδίδουν χρήσιμη διαίσθηση προς την κατανόηση του τρόπου λειτουργίας των μνημών. Είναι σημαντικό να σημειωθεί ότι πολλές από τις πληροφορίες σε αυτήν την ενότητα βασίζονται στην δημοσίευση [30].

2.3.1 Χαρακτηριστικά Επίδοσης

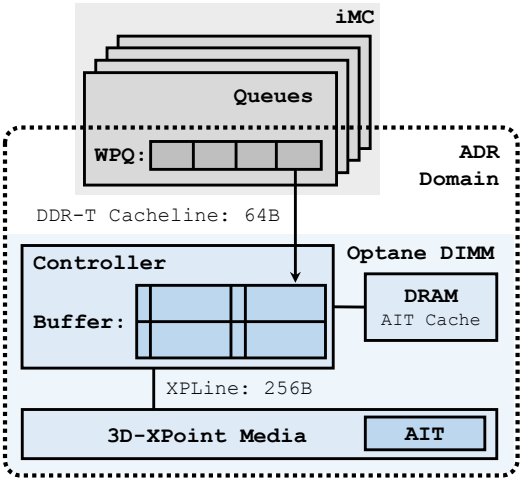
Προς ανάπτυξη των όσων αναφέρθηκαν στην προηγούμενη ενότητα, συγκρινόμενες με παραδοσιακά μέσα αποθήκευσης οι μη πτητικές μνήμες επιτυγχάνουν μεγαλύτερη ρυθμαπόδοση, και μικρότερο λανθάνοντα χρόνο (latency) κατά μια τάξη μεγέθους. Οι μη πτητικές μνήμες υστερούν σε σχέση με την DRAM και προς τις δύο προαναφερθείσες μετρικές [30], προσφέροντας όμως μη πτητικότητα και μεγαλύτερη πυκνότητα. Όπως θα δούμε και στο κεφάλαιο *Πειράματα*, μια σημαντική ιδιορρυθμία των μη πτητικών μνημών αποτελεί η ανομοιομορφία μεταξύ χαρακτηριστικών επίδοσης ανάγνωσης και εγγραφής. Συγκεκριμένα, η ρυθμαπόδοση αναγνώσεων είναι τριπλάσια αυτής των εγγραφών, ενώ στην τεχνολογία DRAM αυτές είναι ομοιόμορφες.

2.3.2 Διεπαφή Επικοινωνίας

Παραδοσιακά αποθηκευτικά μέσα συνδέονται στο υπολογιστικό σύστημα είτε μέσω της διεπαφής SATA, είτε για τα αποδοτικότερα μεταξύ αυτών επί της φυσικής διεπαφής PCI Express μέσω της λογικής διεπαφής NVMe. Οι μη πτητικές μνήμες της Intel συνδέονται επί του διαύλου μνήμης (memory bus) όπως μια μονάδα DRAM. Η διεπαφή επικοινωνίας της μη πτητικής μνήμης με τον iMC (integrated Memory Controller) του επεξεργαστή λέγεται DDR-T, και έχει τα ίδια φυσικά χαρακτηριστικά με αυτά της DDR4, αλλά διαθέτει διαφορετικό πρωτόκολλο επικοινωνίας [5].

2.3.3 Αρχιτεκτονική της διαδρομής επεξεργαστή - μέσου

Η περιγραφή που ακολουθεί βασίζεται στο αμέσως επόμενο σχήμα της δημοσίευσης[30].

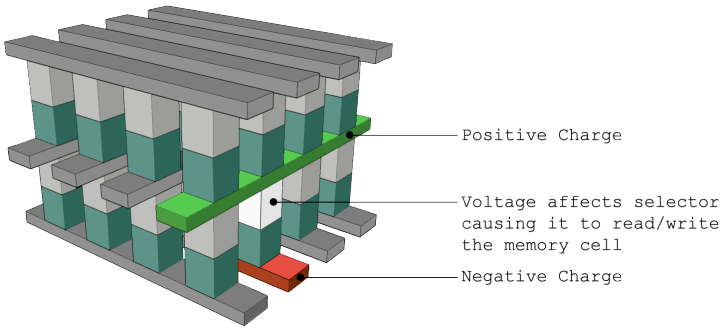


Σχήμα 2.3: Αφαιρετική, συνολική εικόνα της αρχιτεκτονικής για τις μη πτητικές μνήμες

Όπως φαίνεται και στο σχήμα 2.1, κάθε NUMA κόμβος διαθέτει τον δικό του iMC (integrated Memory Controller). Κάθε αίτημα προς τις μνήμες προωθείται από τον επεξεργαστή σε αυτές μέσω του iMC. Αυτός περιλαμβάνει πολλαπλές ουρές ανάγνωσης (RPQ) και εγγραφής (WPQ). Μια κρίσιμη διαφορά των δύο είναι ότι οι ουρές εγγραφής κάθονται στο λεγόμενο Asynchronous DRAM Refresh domain (ADR). Οτιδήποτε στο ADR είναι εξασφαλισμένο ότι θα επιβιώσει σε περίπτωση διακοπής τροφοδοσίας του συστήματος, δηλαδή θα γραφτεί εγκαίρως στις μη πτητικές μνήμες. Ο iMC επικοινωνεί το αίτημα προς την αντίστοιχη μονάδα μη πτητικής μνήμης (Optane DIMM) μέσω της διεπαφής DDR-T με granularity ίσο με ενός cacheline (64 byte).

Για την λειτουργία των Optane DIMM σημειώνουμε δύο σημαντικά πράγματα που αφορούν την τεχνολογία 3D-XPoint: το πρώτο είναι ότι το granularity πρόσβασης στο μέσο είναι 256 byte (το λεγόμενο XPLine στο σχήμα), δηλαδή τέσσερις φορές μεγαλύτερο αυτού της διεπαφής DDR-T. Το δεύτερο είναι ότι όπως συνηθίζεται σε άλλα μέσα αποθήκευσης σαν τους δίσκους SSD, έτσι και στις μη πτητικές μνήμες χρειάζεται μηχανισμός αντιστάθμισης της φθοράς (wear leveling) του μέσου για να αξιοποιηθούν καλύτερα οι αντοχές του.

Απόρροια της διαφοράς στο granularity μεταξύ DDR-T και του μέσου 3D-XPoint αποτελεί εν μέρει η διαμεσολάβηση ενός προσωρινού σημείου αποθήκευσης στον ελεγκτή της συσκευής που συνδυάζει γειτονικές προσβάσεις πριν την εγγραφή στο μέσο (write-combining buffer ή XPBuffer στο σχήμα). Αυτό αποτρέπει την διόγκωση των μικρών εγγραφών. Από την άλλη, λόγω της ανάγκης για wear leveing, καθιερώνεται ένα address indirection table (AIT) και συμπεριλαμβάνεται ένας buffer (AIT Cache) προς εξομάλυνση της λειτουργίας του.



Σχήμα 2.4: Η τεχνολογίας 3D cross-point
Εικόνα βασισμένη σε σχήμα από άρθρο του BBC με τίτλο "3D Xpoint memory: Faster-than-flash storage unveiled"

2.3.4 Τεχνολογία του μέσου

Οι μονάδες μη πτητικών μνημών που αξιοποιήθηκαν για τους σκοπούς αυτής της εργασίας, κοινώς τα Optane DIMM, βασίζονται στην τεχνολογία 3D Xpoint που αποτέλεσε προϊόν συνεργασίας μεταξύ της Intel και της Micron Technology. Διαφοροποιήθηκε από τις έως τότε εκδοχές της τεχνολογίας PCM (Phase Change Memory) προσφέροντας γρηγορότερες και σταθερότερες κυψέλες μνήμης (memory cells) μέσω χρήσης threshold switches αντί για transistor ως επιλογείς, κατασκευασμένους με βάση χαλκογόνα υλικά [4].

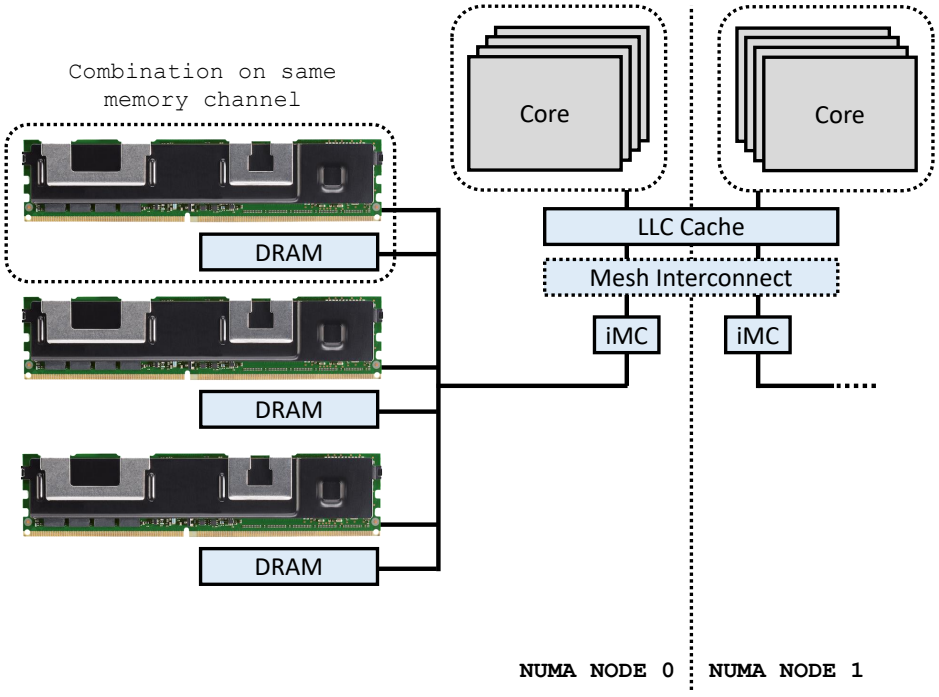
Η επιλογή bit εγγραφής γίνεται μέσω της δομής Cross Point, στην οποία για μια τετράγωνη συστοιχία από bits έχουμε στην μια μεριά (πάνω ή κάτω)

της συστοιχίας καλώδια - στήλες, και στην άλλη μεριά έχουμε καλώδια - γραμμές. Από την ενεργοποίηση ενός καλωδίου στήλης και ενός καλωδίου γραμμής, λαμβάνουμε μια μοναδική κυψέλη στην οποία γίνεται η πρόσβαση, όπως φαίνεται στο σχήμα 2.4. Η δομή αυτή προσφέρει δυνατότητα στοιβασίας (χαρακτηρίζεται ως "stackable"), η οποία προσδίδει στις συσκευές την ιδιότητα της μεγαλύτερης πυκνότητας [3].

2.3.5 Διαμόρφωση των μη πτητικών μνημών

Έχουμε δύο βασικές επιλογές για τον τρόπο διαμόρφωσης των μη πτητικών μνημών. Η πρώτη επιλογή, η οποία είναι και η μόνη που έχει νόημα για χρήση των μνημών ως μέσο αποθήκευσης, είναι το AppDirect mode. Σε αυτό, οι συσκευές αξιοποιούνται ως μέσο αποθήκευσης, και έχουν ξεχωριστή λειτουργία από την DRAM.

Η άλλη επιλογή είναι το memory mode, στο οποίο κάθε μονάδα μη πτητικής μνήμης συνδυάζεται με μια μονάδα μνήμης DRAM στο ίδιο κανάλι μνήμης. Η μη πτητική μνήμη φαίνεται στο λειτουργικό ως κύρια μνήμη, και η DRAM λαμβάνει ρόλο μιας direct-mapped cache της πρώτης. Η λειτουργία της DRAM ως cache γίνεται από τον memory controller του επεξεργαστή [30].

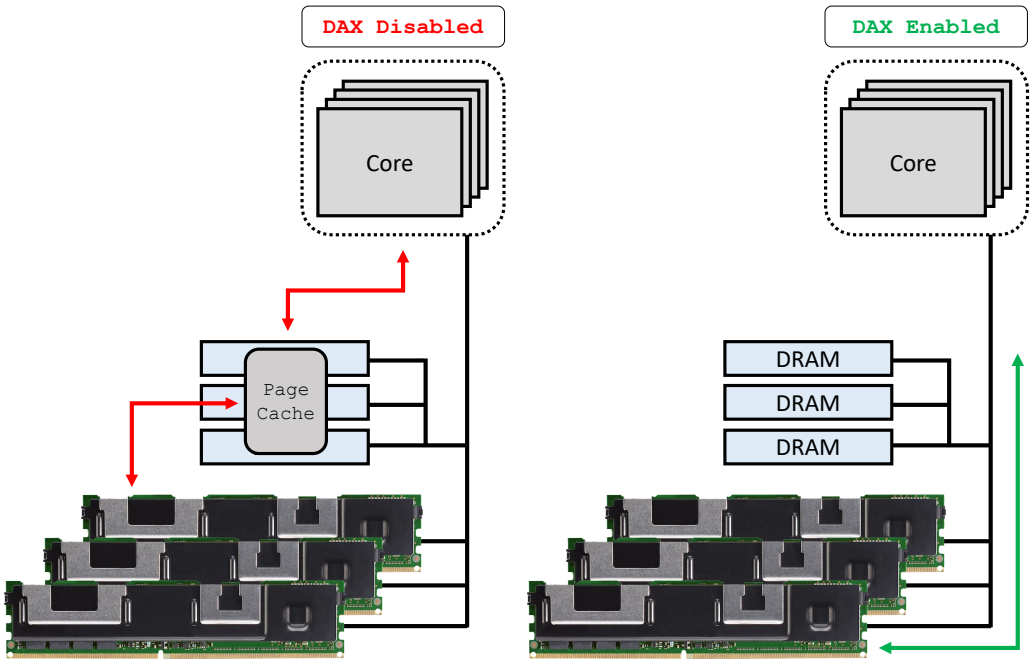


Σχήμα 2.5: Ο τρόπος λειτουργίας memory mode

Ο τρόπος λειτουργίας AppDirect φαίνεται στο σχήμα 2.1, ενώ ο τρόπος λειτουργίας Memory φαίνεται στο σχήμα 2.5.

2.3.6 Λειτουργία DAX σε συστήματα αρχείων

Για μέσα που προσφέρουν γρήγορη, byte adressable διεπαφή όπως οι μη πτητικές μνήμες, σε συστήματα αρχείων του πυρήνα Linux όπως τα ext2, ext4 και xfs ορίζεται η έννοια της λειτουργίας DAX. Όταν η λειτουργία αυτή είναι ενεργή, η πρόσβαση στο συμβατό μέσο αποθήκευσης γίνεται απευθείας, χωρίς μεσολάβηση της page cache. Αντιστοίχως, με απενεργοποιημένο το χαρακτηριστικό DAX οι προσβάσεις δεδομένων γίνονται μέσω της page cache, δηλαδή αποθηκεύονται προσωρινά σελίδες στην DRAM για να είναι γρηγορότερες πιθανές μελλοντικές προσβάσεις. Η page cache χρειάζεται κάποια στιγμή να γράψει πίσω στο μέσο οποιαδήποτε τροποποιημένη σελίδα ώστε να το διατηρεί ενημερωμένο.



Σχήμα 2.6: Ο τρόπος λειτουργίας DAX

Η page cache προσφέρει καλύτερη ρυθμαπόδοση για τις προσβάσεις στις αποθηκευμένες σελίδες της, εφόσον κάθεται στην DRAM. Ωστόσο, η χωρητικότητα της μπορεί να είναι αισθητά μικρότερη από αυτήν του μέσου αποθήκευσης, και για κάθε σελίδα που δεν υπάρχει ήδη στην page cache απαιτούνται δύο αντίγραφα (φαινόμενο double copying), ένα για να μεταφερθεί η σελίδα

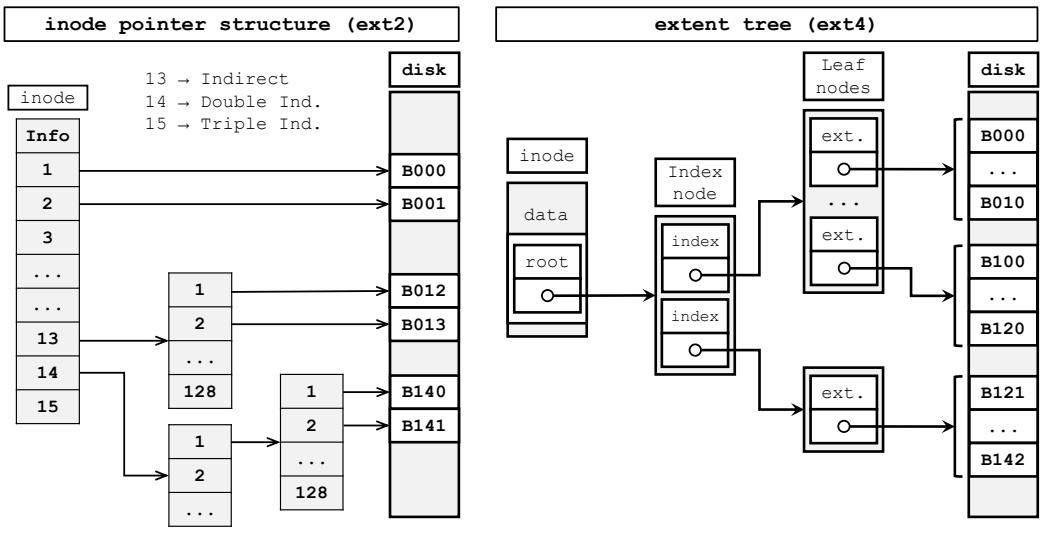
στην page cache, και άλλο ένα για να μεταφερθούν τα δεδομένα στην αιτούμενη διεργασία. Σε μέσα που έχουν συγκρίσιμη επίδοση με την DRAM, όπως οι μη πτητικές μνήμες, μπορεί ανά περιπτώσεις να πλήττονται περισσότερο από το double copying παρά να ευνοούνται από το μεγαλύτερο bandwidth που προσφέρει η DRAM.

2.4 Προϋπάρχουσες βελτιστοποιήσεις του Ext4

Σε αυτήν την ενότητα θα εξετάσουμε μερικές από τις βελτιστοποιήσεις που επιστρατεύει το ext4, και οι οποίες αποδείχθηκαν άμεσα συσχετισμένες με την διαδικασία ανάπτυξης του κώδικα της εργασίας αυτής. Παρόλο που τυπικά ο σκοπός του παρόντος κεφαλαίου είναι να παρουσιαστεί η θεωρητική βάση στην οποία χτίζει η εργασία, χωρίς ακόμα να παρατεθούν λεπτομέρειες για την δουλειά που έγινε, θεωρήθηκε σκόπιμο προς ευκολία αναφοράς για κάθε βελτιστοποίηση να αναφερθεί συνοπτικά η προσαρμογή που έγινε στο κομμάτι της υλοποίησης. Αναλυτικότερη παρουσίαση θα πραγματοποιηθεί στην ενότητα *Τροποποιήσεις στο Ext4* του κεφαλαίου *Υλοποίηση*

2.4.1 Extent Tree

Μια σημαντική σχεδιαστική απόφαση στα συστήματα αρχείων είναι ο τρόπος με τον οποίο τα μεταδεδομένα ενός αρχείου δεικτοδοτούν τα block δεδομένων του.



Σχήμα 2.7: Μέθοδοι δεικτοδότησης δεδομένων αρχείων

Ιστορικά, σε συστήματα αρχείων όπως το ext2 ο τρόπος δεικτοδότησης των block δεδομένων ήταν μέσω πινάκων σταθερού μεγέθους, με κάθε στοιχείο του πίνακα να είναι δείκτης σε κάποιο block ή σε άλλον ίδιου τύπου πίνακα, όπως φαίνεται στα αριστερά του σχήματος 2.7. Στο ext2, τα αρχεία ξεκινούν με έναν πίνακα 15 θέσεων. Η θέση 13 δείχνει σε έναν πίνακα 128 θέσεων που δεικτοδοτεί απευθείας block. Λόγω αυτής της δομής, η θέση 13 λέγεται ότι κάνει (μονή) ανακατεύθυνση.

Η θέση 14 δείχνει σε έναν πίνακα ο οποίος δεικτοδοτεί πίνακες όπως αυτός της θέσης 13. Πλέον δηλαδή έχουμε κάτι που μοιάζει με δομή δέντρου 2 επιπέδων. Η θέση 15, που είναι και η τελευταία του αρχικού πίνακα, δεικτοδοτεί μια παρόμοια δομή, αλλά τριών επιπέδων. Για αυτό οι θέσεις 14 και 15 λέμε αντίστοιχα ότι κάνουν διπλή και τριπλή ανακατεύθυνση.

Όλοι οι πίνακες εκτός του αρχικού έχουν μέγεθος 128 θέσεων και δεσμεύονται μόνο όταν είναι απαραίτητο. Δηλαδή, ο πίνακας της θέσης 13 δεσμεύεται όταν οι υπόλοιπες 12 θέσεις του αρχικού πίνακα έχουν καταληφθεί, ο δεξιότερος πίνακας στο σχήμα 2.7 δεσμεύεται όταν έχουν καταληφθεί όλες οι θέσεις του πίνακα μονής ανακατεύθυνσης, και αφού μόλις έχει δεσμευθεί ο πίνακας της θέσης 14 για να γίνεται η διπλή ανακατεύθυνση κλπ.

Αυτό το σχήμα δεικτοδότησης δεν εκμεταλλεύεται την συχνή περίπτωση στην οποία αρκετά block έχουν δεσμευτεί με διαδοχικό τρόπο πάνω στο μέσο αποθήκευσης, ώστε να αποφευχθεί η σπατάλη χώρου λόγω μεταδεδομένων. Κοινώς, δεικτοδοτώντας εύρη από block αντί για κάθε block μεμονωμένα μπορούμε να έχουμε μια πιο οικονομική αναπαράσταση των block δεδομένων κάθε αρχείου. Αυτήν ακριβώς την ιδέα επιστρατεύει το ext4, το οποίο εισάγει την έννοια του extent tree, μιας δενδρικής δομής με φύλλα που δεικτοδοτούν ένα εύρος από συνεχόμενα block (το λεγόμενο extent). Η δομή αυτή παρουσιάζεται στο δεξί μέρος του σχήματος 2.7.

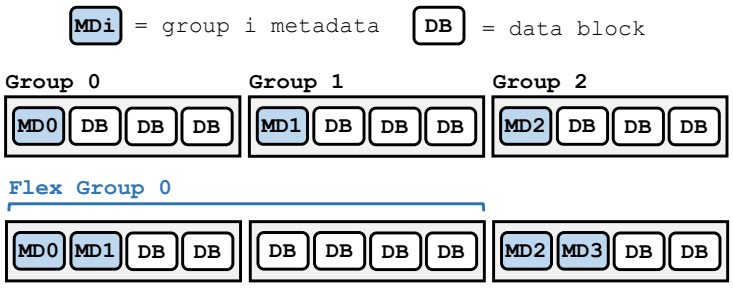
Ενώ σε αυτήν την εργασία δεν θα μεταβάλλουμε την λειτουργία του extent tree με οποιοδήποτε τρόπο, βοηθάει να γνωρίζουμε το σχήμα δεικτοδότησης εφόσον η δομή του με λογικό τρόπο επηρεάζει άμεσα τον τρόπο δέσμευσης καινούργιων block. Έτσι, θα δούμε τι περιορισμοί επιβάλλονται στην κατά NUMA τοπικότητα των δεσμεύσεων λόγω της αλληλεπίδρασης του extent tree με άλλες βελτιστοποιήσεις, και στις επεκτάσεις θα αναφερθεί σύντομα πως θα έπρεπε να προσαρμοστεί η λειτουργία του για να υποστηρίζονται κατά NUMA τοπικά overwrite αρχείων.

2.4.2 Flexible Block Groups

Ορίζουμε ομάδες από block group (τα block group είναι ομάδες block, σταθερού μεγέθους), οι οποίες ομάδες ονομάζονται flexible block groups. Σε κάθε ομάδα, το πρώτο block group συλλέγει τα μεταδεδομένα του ίδιου και των υπο-

λοίπων της ομάδας. Με αυτόν τον τρόπο επιτυγχάνουμε καλύτερη τοπικότητα μεταδεδομένων, και λαμβάνουμε μεγαλύτερα κομμάτια συνεχόμενων block δεδομένων εφόσον δεν παρεμβάλλονται μεταδεδομένα το ίδιο συχνά.

Για λόγους ευκολίας, το χαρακτηριστικό αυτό δεν είναι συμβατό με τις επεκτάσεις για το NUMA awareness προς το παρόν.



Σχήμα 2.8: Flexible Block Groups: Με το `flex_bg` ενεργό, το group 0 συλλέγει τα μεταδεδομένα του ίδιου και του group 1, οπότε δημιουργείται το flex group 0.

2.4.3 Locality Group Preallocations

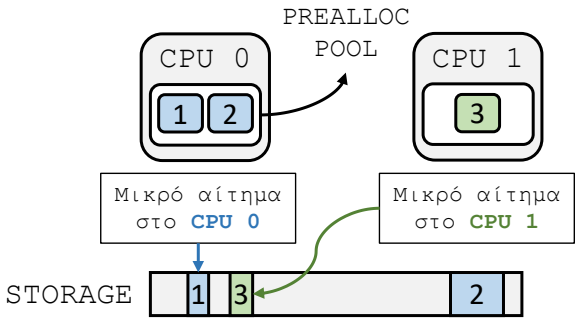
Προσπαθούμε να ικανοποιήσουμε αιτήματα σχετικά μικρού μεγέθους από ένα preallocation pool του αιτούμενου επεξεργαστή. Με αυτόν τον τρόπο επιτυγχάνουμε τοπικότητα και ταχύτητα στην δέσμευση μικρών αρχείων.

Προσαρμόζουμε το χαρακτηριστικό αυτό ώστε να είναι συμβατό με την κατά NUMA τοπικότητα επιλέγοντας από το αντίστοιχο locality group preallocation pool μόνο preallocation που αντιστοιχούν στον τοπικό NUMA κόμβο.

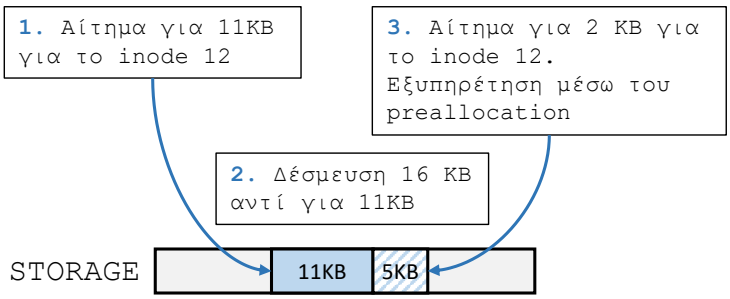
2.4.4 Per-inode Preallocations

Σε κάθε αίτημα για δέσμευση block ενός αρχείου δεσμεύουμε περισσότερο χώρο από τον ζητούμενο και κρατάμε το περίσσευμα ως preallocation για μελλοντική χρήση. Με αυτόν τον τρόπο λαμβάνουμε ιδανικότερου μεγέθους δεσμευμένες περιοχές που συχνά αντιστοιχούν σε επιλεγμένες δυνάμεις του 2, και διευκολύνονται αρκετά οι μελλοντικές δεσμεύσεις.

Προκειμένου η βελτιστοποίηση αυτή να γίνει NUMA aware, για κάθε inode, αντί για ένα μοναδικό preallocation list κρατάμε πλέον από μια ξεχωριστή λίστα για κάθε NUMA κόμβο.



Σχήμα 2.9: Locality Group Preallocations: Συμβολίζουμε τις αναφορές στα preallocation του locality group ως αριθμημένα παραλληλόγραμμα με χρώμα διαφορετικό για κάθε επεξεργαστή. Εδώ φαίνεται ότι κάθε επεξεργαστής συμβουλεύεται το δικό του locality group preallocation space για τις μικρές δεσμεύσεις που εξυπηρετεί.

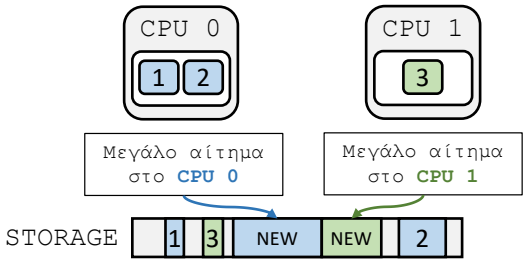


Σχήμα 2.10: Per-inode Preallocations: Παρουσιάζεται η προληπτική δέσμευση χώρου για ένα inode που αξιοποιείται σε επόμενο αίτημα. Το σκιαγραφημένο μέρος εννοείται πως δεικτοδοτείται μέσω μιας λίστας που αντιστοιχεί σε κάποια δομή του inode.

2.4.5 Stream Allocation

Αιτήματα σχετικά μεγάλου μεγέθους δεν αξιοποιούν το locality group preallocation pool, και προσπαθούμε να αρχίσουμε το allocation από το σημείο που ολοκληρώθηκε το προηγούμενο stream allocation. Με αυτόν τον τρόπο λαμβάνουμε πιο ομαλή κίνηση IO, και αποφεύγουμε την δημιουργία μεγάλων fragment (κομμάτι χώρου μεταξύ χωρικά διαδοχικών δεσμεύσεων).

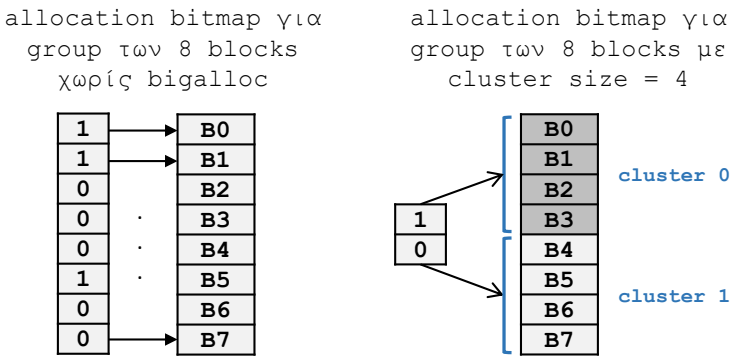
Η προσαρμογή της βελτιστοποίησης για τους σκοπούς του NUMA awareness είναι άμεση, καθώς αρκεί να κρατάμε από ένα global goal (κοινώς το σημείο όπου ολοκληρώθηκε το προηγούμενο stream allocation) για κάθε NUMA κόμβο.



Σχήμα 2.11: Stream Allocation: Θεωρούμε χρονικά ότι προηγείται το αίτημα του CPU 0 έναντι αυτού του CPU 1. Βλέπουμε ότι οι δεσμεύσεις των αιτημάτων αυτών γίνονται με συνεχή τρόπο χωρικά.

2.4.6 Clutser Allocation / Bigalloc

Στο allocation bitmap κάθε group δεν σημειώνουμε μεμονωμένα block, αλλά ομάδες από block προκαθορισμένου μεγέθους, τα λεγόμενα cluster [6]. Ο εναλλακτικός τίτλος προκύπτει από το flag `bigalloc` που μπορεί να αξιοποιηθεί κατά την δημιουργία του συστήματος αρχείων. Με τα cluster λαμβάνουμε μικρότερο overhead και περιορίζεται ο απαιτούμενος χώρος για μεταδεδομένα. Σε κάποιες περιπτώσεις μπορεί να ευνοεί και την τοπικότητα των δεδομένων ενός αρχείου.



Σχήμα 2.12: Cluster Allocation: Βλέπουμε την μείωση της δομής για το allocation bitmap όταν αυξάνουμε το cluster size.

Το χαρακτηριστικό αυτό δεν έχει ληφθεί υπόψη στην υλοποίηση, αλλά περιγράφεται η πιθανή επίδραση του ως προς το NUMA awareness.

Κεφάλαιο 3

Μεθοδολογία

Σε αυτήν την ενότητα θα καλύψουμε την πλειοψηφία των μεθόδων που χρησιμοποιήθηκαν για τους σκοπούς των πειραμάτων και της ανάπτυξης του πυρήνα. Θα ξεκινήσουμε παρουσιάζοντας στις πρώτες δύο ενότητες τα εργαλεία FIO και Filebench, τα οποία χρησιμοποιούμε για εκτέλεση δοκιμών και κυρίως ως μετροπρογράμματα. Στις ακόλουθες ενότητες θα εξετάσουμε την διαμόρφωση κατάλληλης εικονικής μηχανής για τον σκοπό τροποποίησης του πυρήνα, και τον τρόπο εξερεύνησης και αποσφαλμάτωσης του κώδικα του.

3.1 Το Εργαλείο FIO

Το FIO αποτελεί ένα ευέλικτο εργαλείο προσομοίωσης φόρτου εισόδου-εξόδου (IO) για σκοπούς δοκιμών και πραγματοποίησης benchmarking. Γράφτηκε με αφορμή την ανάγκη συστηματικού ελέγχου στην ανάπτυξη του υποσυστήματος I/O και scheduler του πυρήνα Linux [11]. Προσφέρει την δυνατότητα παραμετροποίησης και εκτέλεσης απλών και καλά ορισμένων διεργασιών IO, με καταγραφή αναλυτικών στατιστικών.

Για τους σκοπούς της παρούσας εργασίας, οι δυνατότητες του FIO εξυπηρετούν πρωτίστως την συλλογή των δεδομένων επίδοσης των μη πτητικών συσκευών, όπως θα δούμε στο κεφάλαιο *Πειράματα*. Σε επόμενο στάδιο, το FIO βοηθάει και στους ελέγχους αξιολόγησης ορθότητας και αποδοτικότητας της υποδομής που δημιουργούμε στο κεφάλαιο *Υλοποίηση*.

Το FIO δέχεται ως παράμετρο την περιγραφή μιας εργασίας, η οποία μπορεί να δημιουργεί πολλές διεργασίες του συστήματος, και κάθε διεργασία μπορεί να δημιουργήσει πολλαπλές οντότητες εκτέλεσης (διεργασίες ή νήματα) της δουλειάς που της αντιστοιχεί. Η παραμετροποίηση της εργασίας που εκτελεί το FIO πραγματοποιείται είτε μέσω των παραμέτρων εκτέλεσης του ίδιου του μετροπρογράμματος, είτε μέσω αρχείων εισόδου που ακολουθούν ειδικούς

κανόνες μορφοποίησης [11]. Στον πίνακα 3.1 παρουσιάζουμε κάποιες από τις βασικότερες παραμέτρους που ρυθμίσαμε για τους σκοπούς των μετρήσεων, όπως φαίνονται στα αρχεία του καταλόγου `experiments/dotfio/simple/` του πηγαίου της κώδικα της εργασίας.

Παράμετρος	**Λειτουργία**
`thread`	Δημιουργία οντοτήτων εκτέλεσης των ζητούμενων εργασιών μέσω POSIX thread αντί της fork.
`group_reporting`	Εκτύπωση στατιστικών για όλα τα νήματα εκτέλεσης αντί για το καθένα μεμονωμένα. Σημαντικό για ευκολία στο parsing των αποτελεσμάτων.
`ioengine`	Μέσω ποιας διεπαφής πραγματοποιείται το I/O σε κάθε αρχείο. Για τους σκοπούς της εργασίας, το θέτουμε ίσο με sync, οπότε έχουμε χρήση των read και write syscall. Ο πειραματισμός με memory mapped files είναι εφικτός επιλέγοντας τιμή mmap για αυτήν την παράμετρο.
`size`	Ορισμός μεγέθους αρχείου για κάθε νήμα. Για τους σκοπούς αυτής της εργασίας, τυπικά είναι ίσο με 256MB ή 4GB.
`bs`	Το μέγεθος του block (κοινώς του buffer) που χρησιμοποιούμε ως παράμετρο στις επιμέρους κλήσεις συστήματος. Διαλέξαμε την προεπιλεγμένη τιμή των 4KB.
`rw`	Είδος του I/O μοτίβου. Το θέτουμε `randrw` ώστε να μπορούμε να έχουμε μίξη εγγραφών και αναγνώσεων μέσω της παραμέτρου rwmixread, και αν θέλουμε να πειραματιστούμε με τον βαθμό τυχαιότητας του I/O μέσω της παραμέτρου `percentage_random`.
`rwmixread`	Το ποσοστό των προσβάσεων μνήμης που είναι αναγνώσεις αντί για εγγραφές.
`percentage_random`	Βαθμός τυχαιότητας του IO. Για τιμή της παραμέτρου ίση με 0 έχουμε ακολουθιακό IO, ενώ για τιμή ίση με 100 έχουμε προσβάσεις σε εντελώς τυχαία σημεία του αρχείου. Για τους σκοπούς αυτής της εργασίας το θέτουμε ίσο με 0.
`invalidate`	Ακύρωση των αποθηκευμένων σελίδων στην page cache των αρχείων που χρησιμοποιεί η ορισμένη εργασία. Έχει νόημα μόνο αν το DAX είναι απενεργοποιημένο.

`create_ serialize`	Αν η παράμετρος είναι ίση με 1, η δημιουργία των αρχείων κάθε νήματος δεν γίνεται από τα ίδια τα νήματα, αλλά από τον πατέρα τους, πριν την δημιουργία των νημάτων–παιδιών. Διαφορετικά, κάθε νήμα δημιουργεί το δικό του αρχείο όταν ξεκινήσει την εκτέλεση του. Ιδιαίτερα χρήσιμη παράμετρος όταν θα θελήσουμε στην αξιολόγηση της υλοποίησης να εξετάσουμε αν η δέσμευση χώρου για τα αρχεία σέβεται την τοπικότητα κατά NUMA των νημάτων που τα δημιουργούν.
`loops`	Αριθμός επαναλήψεων της ορισμένης εργασίας. Τυπικά ίσος με 1. Χρήσιμη ως παράμετρος όταν θέλουμε να εξετάσουμε πως με το DAX απενεργοποιημένο η ύπαρξη σελίδων στην page cache μετά την πρώτη επανάληψη μπορεί να οδηγήσει σε ταχύτερη διεκπεραίωση του ζητούμενου IO.
`numjobs`	Πόσες οντότητες εκτέλεσης θα δημιουργηθούν για την ζητούμενη εργασία (job).
`directory`	Το directory στο οποίο βρίσκονται ή τοποθετούνται τα αρχεία κάθε οντότητας εκτέλεσης της εργασίας. Μέσω αυτής της παραμέτρου οδηγούμε το FIO στην χρήση των μη πτητικών μηνών, παρέχοντας κάποιο directory στο αντίστοιχο mountpoint.
`cpus_allowed`	Λίστα με τους πυρήνες του επεξεργαστή στους οποίους θέλουμε να περιορίσουμε την εκτέλεση αυτής της εργασίας. Με αυτήν την παράμετρο επιτυγχάνουμε την εκτέλεση της εργασίας σε επιλεγμένο NUMA κόμβο.

Πίνακας 3.1: Οι βασικές παράμετροι λειτουργίας του FIO που αξιοποιούνται στην παρούσα εργασία

Προκειμένου να δημιουργήσουμε εργασίες με νήματα που είναι κατανεμημένα μεταξύ NUMA κόμβων, ορίζουμε από μια ξεχωριστή διεργασία για τον κάθε κόμβο. Επειδή οι διεργασίες αυτές έχουν πολλές κοινές παραμέτρους, τις συλλέγουμε στο αρχείο `experiments/dotfio/simple/global.fio`. Οι παράμετροι που διαφοροποιούν τις επιμέρους διεργασίες φαίνονται κάτω από την διπλή γραμμή του πίνακα 3.1, και ρυθμίζονται με τον τρόπο που φαίνεται για παράδειγμα στο αρχείο `experiments/dotfio/simple/mixed.fio`.

Συνοψίζοντας, στο αρχείο `experiments/instances/fio_instance.sh` του πηγαίου κώδικα της εργασίας φαίνεται αναλυτικά ο τρόπος κλήσης του

FIO και ρύθμισης των παραμέτρων που έχουμε αφήσει μεταβλητές στα αρχεία με κατάληξη `.fio`. Ο τρόπος καθορισμού της παραμέτρου `cpus_allowed` θα αιτιολογηθεί στην ενότητα *Μεθοδολογία Συλλογής Μετρήσεων*.

3.2 Το Εργαλείο Filebench

Το Filebench είναι ένα μετροπρόγραμμα για συστήματα αρχείων και τεχνολογίες μέσων αποθήκευσης [27]. Διαφοροποιείται από το FIO στο γεγονός ότι η κίνηση εισόδου-εξόδου που παράγει μπορεί να ακολουθήσει πιο περίπλοκα μοτίβα, με τρόπο ικανό να προσομοιώνει την συμπεριφορά διάφορων εφαρμογών υπολογιστικών συστημάτων. Αυτό επιτυγχάνεται μέσω μιας δικής του γλώσσας περιγραφής εργασιών, που ονομάζεται Workload Model Language.

Τον πηγαίο κώδικα του Filebench συνοδεύουν διάφορα προκαθορισμένα σενάρια εκτέλεσης (workloads) τα οποία χρησιμοποιούνται αρκετά στην βιβλιογραφία [18, 28, 13]. Σε αυτήν την εργασία θα αξιοποιήσουμε το Filebench για να αξιολογήσουμε την ορθότητα της δουλειάς της ενότητας *Υλοποίηση*, καθώς και το κέρδος στην επίδοση που αυτή επιφέρει. Από τα προκαθορισμένα σενάρια εκτέλεσης, αξιοποιούμε με κατάλληλες προσαρμογές αυτά του πίνακα 3.2.

Workload	**Περιγραφή**
`fileserver`	Προσομοιώνει την συμπεριφορά ενός διακομιστή αρχείων, ο οποίος γράφει καινούργια αρχεία, επεκτείνει, διαγράφει, διαβάζει ή ζητάει πληροφορίες (stat) για ήδη υπάρχοντα αρχεία.
`varmail`	Προσομοιώνει την συμπεριφορά χρήστη ενός mail server, ο οποίος διαγράφει, δημιουργεί, απαντάει σε και διαβάζει μηνύματα. Θεωρεί ότι τα μηνύματα έχουν κατά μέσο όρο μέγεθος 16KB.
`webserver`	Προσομοιώνει την συμπεριφορά ενός διακομιστή δικτύου, που κατά κύριο λόγο διαβάζει αρχεία που του ζητούνται, και παροδικά κάνει επέκταση ενός αρχείου καταγραφής (log).

Πίνακας 3.2: Τα workload του Filebench που αξιοποιούνται στην παρούσα εργασία

Σημειώνεται ότι για τους σκοπούς αυτής της εργασίας δεν χρησιμοποιούμε το Filebench ως έχει (δηλαδή στην κατάσταση του commit με πρόθεμα 22620e6 στο GitHub repository του εργαλείου), αλλά το επεκτείνουμε με κατάλληλο για

κάθε περίπτωση τρόπο. Από αυτήν την ανάγκη προέκυψαν τα αρχεία του καταλόγου `experiments/patches` του πηγαίου κώδικα. Οι επεκτάσεις που έγιναν στο Filebench θα περιγραφούν αναλυτικότερα στην ενότητα *Έλεγχος Αποδοτικότητας μέσω του Filebench* του κεφαλαίου *Αξιολόγηση*.

3.3 Μεθοδολογία Συλλογής Μετρήσεων

Προκειμένου να μπορούμε με ασφάλεια να εξάγουμε συμπεράσματα από τις μετρήσεις που πραγματοποιούμε σε ένα σύστημα, ιδίως στην περίπτωση των μη πτητικών μνημών οι οποίες έχουν ιδιαιτερότητες όπως θα δούμε στο κεφάλαιο *Πειράματα*, χρειάζεται να κάνουμε συστηματική την διαδικασία συλλογής μετρήσεων. Σε αυτήν την ενότητα θα δούμε πως να περιορίσουμε τον θόρυβο στις μετρήσεις, και πως αυτοματοποιήθηκε η διαδικασία συλλογής μετρήσεων ώστε να διευκολυνθεί η ροή της εργασίας.

3.3.1 Περιορισμός θορύβου στις μετρήσεις

Όπως αναφέρθηκε προηγουμένως, το FIO αποτελεί βασικό μέσο για την δημιουργία και καταγραφή στατιστικών κίνησης δεδομένων από και προς τις μη πτητικές μνήμες του συστήματος. Μέσω της εισόδου του μετροπρογράμματος αυτού, μπορούμε να διευκρινίσουμε εύκολα και με επακριβή τρόπο τον αριθμό των νημάτων που θα εκτελέσουν τις ζητούμενες προσβάσεις, προς ποια αρχεία, το πως αλληλεπιδρούν τα νήματα με το σύστημα αρχείων και πολλές ακόμα παραμέτρους. Ωστόσο, όσο ακριβείς και αν είμαστε στην περιγραφή του πειράματος, οφείλουμε πριν την εκτέλεση τους να ελαχιστοποιήσουμε πρώτα τους παράγοντες του συστήματος οι οποίοι μπορούν να προσθέσουν θόρυβο στις μετρήσεις.

Πρώτο και σημαντικότερο σημείο εισαγωγής θορύβου είναι ο ίδιος ο επεξεργαστής. Αναλυτικότερα, διακρίνουμε τις παραμέτρους που παρουσιάζονται αμέσως μετά.

Αποφυγή Υπερνημάτωσης

Η υπερνημάτωση θεωρούμε ότι μπορεί να οδηγήσει σε διακυμάνσεις στις μετρήσεις. Ενδέχεται από την μια μέτρηση στην επόμενη να αυξάνεται το ποσοστό χρόνου κατά το οποίο νήματα συνυπάρχουν σε έναν επεξεργαστικό πυρήνα, λόγω ξένων ως προς το πείραμα νημάτων που δρομολογούνται ταυτόχρονα στον επεξεργαστή. Δηλαδή, προτιμάμε όταν είναι εφικτό το κάθε νήμα να μην τοποθετείται με κάποιο άλλο (I/O intensive) νήμα στον ίδιο επεξεργαστικό πυρήνα, αλλά το κατά πόσο αυτό γίνεται σε μια εκτέλεση είναι πολύ ευμετάβλητο. Ο λόγος της προτίμησης αυτής άπτεται στην υπόθεση ότι αξιοποιούνται

καλύτερα έτσι οι δομές προσωρινής αποθήκευσης (cache) των επεξεργαστικών πυρήνων. Κατά συνέπεια, για κάθε επεξεργαστικό πυρήνα αξιοποιούμε το πολύ έναν λογικό. Αυτό είναι κάτι που ρυθμίζεται μέσω της παραμέτρου `cpus_allowed` του FIO.

Χρήση τόσων πυρήνων όσοι είναι απαραίτητοι

Η επαναδρομολόγηση νημάτων σε καινούργιο επεξεργαστικό πυρήνα επιβαρύνει την τοπικότητα δεδομένων στις μνήμες προσωρινής αποθήκευσης. Αυτό μπορούμε να το περιορίσουμε σαν φαινόμενο αν ορίσουμε κατάλληλη μάσκα επεξεργαστών στους οποίους μπορεί να εκτελεστεί το πείραμα. Κοινώς, φροντίζουμε να χρησιμοποιούμε ένα σταθερό σύνολο επεξεργαστικών πυρήνων, πλήθους ισάριθμου των νημάτων του εκάστοτε πειράματος. Αυτό το επιτυγχάνουμε είτε μέσω του προγράμματος `taskset`, είτε με την παράμετρο `cpus_allowed` του FIO.

Αν θέλαμε να περιορίσουμε ακόμα περισσότερο το φαινόμενο που περιγράψαμε, θα μπορούσαμε να αξιοποιήσουμε την παράμετρο `cpus_allowed_policy=split` του FIO, με την οποία καταφέρνουμε κάθε νήμα να εκτελείται σε έναν και μόνο επεξεργαστικό πυρήνα καθ' όλη την διάρκεια της ζωής του. Αυτό όμως από την μια κάνει το πείραμα μας ακόμα λιγότερο ρεαλιστικό, και από την άλλη οδηγεί σε υποχρησιμοποίηση του συστήματος όταν έχουμε περισσότερα νήματα από τους επεξεργαστικούς πυρήνες που διαθέτουμε. Αυτό συμβαίνει καθώς, αν για παράδειγμα έχουμε 17 νήματα που τρέχουν σε 16 πυρήνες, τότε δύο τουλάχιστον νήματα θα ανταγωνίζονται για έναν πυρήνα την στιγμή που μπορεί να υπάρχει κάποιος άλλος διαθέσιμος την δεδομένη χρονική στιγμή. Ο τελικός χρόνος του πειράματος θα είναι σχεδόν διπλάσιος σε σχέση με το αν επιτρέπαμε εκτέλεση σε οποιονδήποτε επιτρεπτό πυρήνα, και λαμβάνουμε φαινομενική ασυνέχεια στις μετρήσεις επίδοσης μεταξύ 16 και 17 νημάτων.

Χρήση στατικής συχνότητας του επεξεργαστή

Η δυναμική ρύθμιση της συχνότητας του επεξεργαστή οδηγεί με εμφανή τρόπο σε διαφοροποιήσεις μεταξύ δειγμάτων των μετρήσεων. Για να σιγουρευτούμε ότι δεν θα εισαχθεί θόρυβος λόγω αυξομείωσης της συχνότητας, απενεργοποιούμε το χαρακτηριστικό αυτό. Προς αυτήν την κατεύθυνση, χρησιμοποιούμε το εκτελέσιμο `cpupower` που μπορεί να παραχθεί στον κατάλογο `tools/power/cpupower` του πηγαίου κώδικα του πυρήνα, και για να είμαστε σίγουροι ενημερώνουμε κατάλληλα το αρχείο `/etc/default/cpufrequtils` επανεκκινώντας την υπηρεσία cpufrqutils του συστήματος, όπως φαίνεται στην συνέχεια.

```
echo 'GOVERNER="performance"' | tee /etc/default/cpufrequtils
systemctl restart cpufrequtils

export LD_LIBRARY_PATH="$cpupowerdir"
$cpupowerdir/cpupower frequency-set -g performance
```

Απενεργοποίηση των idle state του επεξεργαστή

Με τον όρο idle state αναφερόμαστε σε μια κατάσταση ενός επεξεργαστικού πυρήνα κατά την οποία περιορίζονται προσωρινά οι διαθέσιμες λειτουργίες του προς εξοικονόμηση ενέργειας. Όπως στην περίπτωση της δυναμικής ρύθμισης της συχνότητας, γίνεται και εδώ αντιληπτό ότι η απενεργοποίηση του χαρακτηριστικού μπορεί να βοηθήσει προς την ομοιομορφία μεταξύ δειγμάτων και την καλύτερη σύγκριση των μετρήσεων. Επιτυγχάνουμε την ρύθμιση με παρόμοιο τρόπο με αυτόν για την απενεργοποίηση της δυναμικής συχνότητας:

```
systemctl disable ondemand
systemctl restart cpufrequtils

export LD_LIBRARY_PATH="$cpupowerdir"
$cpupowerdir/cpupower idle-set --disable-by-latency 0
```

Επιπροσθέτως, είναι σημαντικό η εκτέλεση των πειραμάτων να γίνεται σε κατάσταση απουσίας άλλων χρηστών του συστήματος που απαιτούν αξιοσημείωτους πόρους από αυτό. Αυτό είναι απαραίτητο γιατί η ένταση της αλληλεπίδρασης του άλλου χρήστη με το σύστημα μπορεί να είναι τυχαία και να επιβαρύνει ανομοιόμορφα την εκτέλεση διαφορετικών μετρήσεων.

Τέλος, μεταξύ επαναλήψεων ενός πειράματος επιθυμούμε να έχουμε όσο το δυνατόν πανομοιότυπες αρχικές συνθήκες. Για παράδειγμα, δεν μπορεί το αξιοποιούμενο σύστημα αρχείων να διαθέτει περιορισμένη χωρητικότητα λόγω προηγούμενου πειράματος, γιατί τότε αλλάζει ο τρόπος αξιοποίησης του μέσου αποθήκευσης στην νέα εκτέλεση.

3.3.2 Αυτοματοποιημένο Σύστημα Διεξαγωγής Πειραμάτων

Όλα όσα αναφέραμε αποτελούν αρκετά βήματα προς έλεγχο, που αν γίνουν χειροκίνητα είναι από την μια πιθανό να γίνουν λάθη στην διαδικασία λόγω του ανθρώπινου παράγοντα. Από την άλλη τα βήματα αυτά απαιτούν αρκετό χρόνο από τον χρήστη, ο οποίος μπορεί να αξιοποιηθεί καλύτερα. Για αυτό τον λόγο, μέσω bash scripting αυτοματοποιούμε πλήρως την διαδικασία εκτέλεσης των πειραμάτων, συλλογής των δεδομένων και οπτικοποίησης των σχετικών

αποτελεσμάτων. Όλα τα απαραίτητα αρχεία που εξυπηρετούν αυτόν τον σκοπό βρίσκονται στο κατάλογο `experiments` του πηγαίου κώδικα που συνοδεύει την εργασία. Αναφέρουμε συνοπτικά τους υποκαταλόγους που περιέχει, και τον ρόλο του καθενός:

- `dotfio/simple`: Σε αυτόν τον κατάλογο βρίσκονται όλα τα αρχεία περιγραφής εργασιών του FIO που αξιοποιούνται για τα πειράματα. Χρειάζεται διάκριση περιπτώσεων αναλόγως με την κατανομή των νημάτων μεταξύ κόμβων, και για αυτό έχουμε πολλαπλά αρχεία. Έχουμε έναν υποκατάλογο του `dotfio` με όνομα `simple` γιατί μπορούμε σε επέκταση της εργασίας να εισάγουμε και άλλα σενάρια χρήσης του FIO προς εξέταση. Για παράδειγμα, μπορούμε να προσθέσουμε έναν φάκελο thinktime, στα αρχεία του οποίου να χρησιμοποιείται η ομώνυμη παράμετρος του FIO που προσομοιώνει περαιτέρω αξιοποίηση των πόρων του επεξεργαστή για να έχουμε νήματα έντασης I/O αλλά και CPU.

- `graphs/scripts`: Σε αυτόν τον κατάλογο έχουμε αρχεία της python που μπορούν να παράξουν πολλά από τα διαγράμματα της εργασίας άμεσα. Χρησιμοποιούμε για αυτό μεταξύ άλλων τις βιβλιοθήκες matplotlib και seaborn. Οι εικόνες που προκύπτουν τοποθετούνται στον κατάλογο `graphs`.

- `instances`: Περιέχει αρχεία παραμετρικής εκτέλεσης των FIO και Filebench. Στο αρχείο `fio_instance.sh` για παράδειγμα εμπεριέχεται η λογική διαμόρφωσης CPU μάσκας, επιλογής του σωστού αρχείου του καταλόγου `dotfio/simple`, λήψης πολλαπλών δειγμάτων ανά μέτρηση, καταγραφής μετρήσεων κλπ.

- `measurements`: Περιέχει κάποιες μετρήσεις που έγιναν, μεταξύ των οποίων και ορισμένες στα συστήματα αρχείων NOVA και WineFS για σκοπούς διερεύνησης.

- `patches`: Περιέχει επεκτάσεις και τροποποιήσεις του κώδικα του Filebench, τα οποία όμως θα μας απασχολήσουν περισσότερο στην ενότητα *Αξιολόγηση*.

- `scenarios`: Σε κάθε αρχείο αυτού του καταλόγου περιέχεται πλήρης περιγραφή πειραμάτων της εργασίας. Αξιοποιούν ό,τι είναι διαθέσιμο στους υπόλοιπους καταλόγους που παραθέτουμε, αυτοματοποιώντας όλη την διαδικασία από την συλλογή μετρήσεων μέχρι την παραγωγή των διαγραμμάτων. Βασιζόμενοι σε αυτά τα αρχεία μπορούμε να γράψουμε με ελάχιστη προσπάθεια καινούργια πειράματα.

- `scripts`: Εδώ βρίσκονται όλα τα bash scripts που σχετίζονται με τις ενέργειες που χρειαζόμαστε για τα πειράματα. Επιγραμματικά, αναφέρουμε ως παραδείγματα τον υποκατάλογο `freq` που προσφέρει το μέσο για να μειώσουμε τον θόρυβο από τον επεξεργαστή και τον υποκατάλογο `mount` που αναλαμβάνει την προσάρτηση συσκευών και την δημιουργία συστημάτων αρχείων επί αυτών.

Αν ο αναγνώστης επιθυμεί να αξιοποιήσει την υποδομή αυτή, αναφέρεται ότι είναι απαραίτητο να χρησιμοποιηθεί το Bash shell λόγω του σχεδιασμού της. Τέλος, θέλει προσοχή η παράμετρος `create_serialize` του αρχείου `experiments/dotfio/simple/global.fio`, η οποία αφέθηκε να ρυθμίζεται χειροκίνητα. Για τους σκοπούς του κεφαλαίου *Πειράματα*, προτιμάμε την τιμή 1, αφού μειώνεται ο θόρυβος στις μετρήσεις λόγω concurrency κατά την δημιουργία των αρχείων σε επίπεδο συστήματος αρχείων. Στο κεφάλαιο *Αξιολόγηση* όμως, και συγκεκριμένα στην ενότητα *Αξιολόγηση Ορθότητας*, θα δούμε ότι για να γίνουν οι δεσμεύσεις αρχείων τοπικά για κάθε νήμα πρέπει να θέσουμε την `create_serialize` ίση με 0.

3.4 Διαμόρφωση Εικονικής Μηχανής

Ως υποδομή για την τροποποίηση και την επακόλουθη αξιολόγηση του πυρήνα Linux, αξιοποιούμε τον προσομοιωτή QEMU (Quick Emulator) σε συνδυασμό με το module εικονικοποίησης KVM (Kernel-based Virtual Machine). Ο προσομοιωτής είναι αυτός που μας επιτρέπει τον προσδιορισμό των χαρακτηριστικών και την εκτέλεση της εικονικής μηχανής που θα χρησιμοποιήσουμε για την ανάπτυξη του πυρήνα. Το KVM προσφέρει στον προσομοιωτή την δυνατότητα χρησιμοποίησης του πυρήνα του ξενιστή λειτουργικού ως hypervisor, και κατ' επέκταση επιτρέπει την αξιοποίηση πιθανών επεκτάσεων εικονικοποίησης σε επίπεδο υλικού (hardware virtualization extensions) του υποκείμενου επεξεργαστή.

3.4.1 Προσδιορισμός των χαρακτηριστικών της εικονικής μηχανής

Πρώτο μας βήμα για την καθιέρωση της ζητούμενης υποδομής ανάπτυξης του πυρήνα αποτελεί η διαμόρφωση μιας εικονικής μηχανής που 1) προσομοιώνει ένα σύστημα NUMA, περιγράφοντας τις συσκευές που είναι προσαρτημένες σε κάθε κόμβο, και 2) προσομοιώνει συσκευές μη πτητικών μνημών.

Για το (1), αρκεί να ορίσουμε τους κόμβους και τις προσαρτημένες πτητικές μνήμες ως παραμέτρους του QEMU με τρόπο παρόμοιο του ακόλουθου:

```
-object memory-backend-ram,size=2G,id=m0
-object memory-backend-ram,size=2G,id=m1

-smp 4,sockets=2,maxcpus=4
-numa node,cpus=0-1,memdev=m0,nodeid=0
-numa node,cpus=2-3,memdev=m1,nodeid=1
```

Στον ορισμό των μη πτητικών μνημών πρέπει να προσδιορίσουμε τον κόμβο στον οποίο προσαρτούμε την καθεμία μέσω της παραμέτρου `node`, όπως θα παρουσιαστεί στην συνέχεια.

Για το (2) σε πρώτο στάδιο ενεργοποιούμε την δυνατότητα υποστήριξης εικονικών μη πτητικών συσκευών από την μεριά του QEMU:

```
-machine pc,nvdimm=on
-enable-kvm
```

Μετά, ορίζουμε κάθε συσκευή όπως φαίνεται στο εξής παράδειγμα:

```
-object memory-backend-file,id=nvm0,share=on,
    ↪ mem-path=nvdimm/nvdimm0,size=4G
-device nvdimm,id=nvdimm0,memdev=nvm0,node=0
```

Συγκεκριμένα, προσδιορίζουμε ένα backend file που δημιουργείται στον ξενιστή, με δοσμένο το directory του και το μέγεθος του. Ύστερα, δημιουργούμε μια συσκευή nvdimm βάσει αυτού του backend αρχείου. Η επιλογή `share=on` εξασφαλίζει ότι το backend file είναι διαμοιραζόμενο μεταξύ host και guest, οπότε μεταξύ κύκλων λειτουργίας της εικονικής μηχανής θα παραμένει ως έχει η διαμόρφωση των εικονικών μη πτητικών συσκευών (όπως και τα αρχεία τους).

Φτιάχνουμε από μια συσκευή για κάθε κόμβο. Ιδανικά, θα θέλαμε να έχουμε από δύο και περισσότερες συσκευές σε κάθε κόμβο, ώστε να πειραματιστούμε με την δημιουργία interleaved namespace στον καθένα. Ωστόσο, ο τρόπος με τον οποίον αυτό επιτυγχάνεται, δηλαδή η δημιουργία ενιαίου memory allocation goal ανά κόμβο όπως φαίνεται στο τέλος της ιστοσελίδας [15], είναι μέσω του εργαλείου IPMCTL της intel. Αυτό με την σειρά του συνεργάζεται με το firmware των συσκευών, και κατά συνέπεια εξαρτάται αρκετά από αυτό. Προφανώς, σε επίπεδο εικονικοποιημένης μη πτητικής συσκευής δεν είναι εφικτή η επιτυχής εκτέλεση του εργαλείου για τον σκοπό που αναφέρθηκε. Συνεπώς, περιοριζόμαστε για την εικονικοποίηση σε μια μόνο συσκευή ανά κόμβο.

Όσα αναφέραμε έως τώρα καλύπτουν τα βασικότερα στοιχεία διαμόρφωσης της εικονικής μηχανής. Περισσότερες λεπτομέρειες, όπως οι παράμετροι για την αρχική εγκατάσταση μιας διανομής Linux και την δικτύωση της μηχανής, φαίνονται στο αρχείο `various/vm.sh` του πηγαίου κώδικα της εργασίας.

Σημειώνεται τέλος ότι σκοπός μας με την διαμόρφωση που περιγράφηκε δεν

είναι η δημιουργία μιας εικονικής μηχανής που προσεγγίζει ποιοτικά τα χαρακτηριστικά επίδοσης ενός αντίστοιχου πραγματικού συστήματος. Επί παραδείγματι, δεν μας ενδιαφέρει να προσομοιώσουμε τις καθυστερήσεις λόγω του δικτύου διασύνδεσης σε επίπεδο NUMA, ούτε να εφαρμόσουμε κάποια μεθοδολογία προσέγγισης του bandwidth των μη πτητικών συσκευών μέσω της DRAM του ξενιστή (όπως συνηθιζόταν να γίνεται σε ερευνητικό επίπεδο προτού γίνουν εμπορικά διαθέσιμες οι μη πτητικές συσκευές της Intel [30]).

Στόχος μας είναι να έχουμε ένα εικονικό σύστημα το οποίο θα φαίνεται σαν ένα αντίστοιχο πραγματικό στον πυρήνα, αν εξαιρέσουμε την λειτουργία των οδηγών εικονικοποίησης των συσκευών που λείπουν από την εικονική μηχανή. Έτσι, μπορούμε να προσδιορίσουμε με ευρετικές μεθόδους στην εικονική μηχανή το μέρος του κώδικα που αφορά για παράδειγμα την λειτουργία DAX και τον (αφαιρετικό) NVDIMM driver, να πραγματοποιήσουμε αλλαγές σε αυτόν, και να ελέγξουμε την ορθή λειτουργία (όχι όμως απαραίτητα την επίδοση) των προαναφερθέντων αλλαγών.

3.4.2 Εγκατάσταση Διανομής Linux

Επιλέχτηκε ως διανομή η Debian Linux. Κατεβάζουμε το αρχείο εικονικού δίσκου για την εγκατάσταση και δημιουργούμε τον εικονικό χώρο αποθήκευσης με μορφοποίηση qcow2 μέσω της επόμενης εντολής:

```
$ qemu-img create -f qcow2 debian.img 25G
```

Προτού εκκινήσουμε την εικονική μηχανή για πρώτη φορά, κάνουμε διαθέσιμες τις επόμενες δύο παραμέτρους του QEMU, ώστε να πραγματοποιήσουμε εκκίνηση από τον εικονικό δίσκο (ISO image) και να προβούμε στην εγκατάσταση:

```
-cdrom <ISO-image> # In this case: debian-11.5.0-amd64-DVD-1.iso
-boot d
```

Εκτελούμε το script της εικονικής μηχανής και ακολουθούμε την διαδικασία μιας τυπικής εγκατάστασης της επιλεγμένης διανομής Linux. Η μόνη πιθανή διαφοροποίηση από τις προεπιλεγμένες ρυθμίσεις κατά την διαδικασία αυτή είναι η παράκαμψη εγκατάστασης γραφικού περιβάλλοντος (εν προκειμένω ήταν προεπιλεγμένο το γραφικό περιβάλλον GNOME). Αυτό είναι επιθυμητό γιατί η συμπερίληψη γραφικού περιβάλλοντος επιβαρύνει σημαντικά την ίδια την εγκατάσταση και θα καθυστερεί την εκκίνηση της εικονικής μηχανής. Αντί αυτού, ενεργοποιούμε την επιλογή "SSH server," ώστε να μπορούμε να συνδεόμαστε στην εικονική μηχανή από έναν προσομοιωτή τερματικού του ξενιστή.

Αυτό είναι προτιμότερο από το να χρησιμοποιούμε π.χ. την γραφική διεπαφή που παρέχει το QEMU για την αλληλεπίδραση με την εικονική μηχανή, αφού αυτή έχει περιορισμένες δυνατότητες (και προϋποθέτει ότι χρησιμοποιούμε τον ξενιστή μέσω κάποιου γραφικού περιβάλλοντος).

Αφού ολοκληρωθεί η εγκατάσταση, αφαιρούμε τις παραμέτρους του script της εικονικής μηχανής που αφορούν την εγκατάσταση μέσω του εικονικού δίσκου, και εκκινούμε την μηχανή για να ρυθμίσουμε την καινούργια εγκατάσταση της διανομής που επιλέξαμε. Αυτό δεν μπορούμε να το κάνουμε μέσω SSH, καθώς ο χρήστης που δημιουργήσαμε κατά την εγκατάσταση (αυτός που δεν είναι ο διαχειριστής) δεν ανήκει στο sudoers group, και κατά συνέπεια δεν μπορούμε μέσω αυτού να κάνουμε ουσιαστικές τροποποιήσεις στο σύστημα. Προσωρινά αρκούμαστε είτε στην γραφική διεπαφή του QEMU, είτε σε κάποια επιλογή του όπως η `-nographic`. Πραγματοποιούμε σύνδεση μέσω του χρήστη root. Τα βήματα που ακολουθούμε για την ολοκλήρωση της ρύθμισης της εικονικής μηχανής, είναι κατά προσέγγιση τα επόμενα:

```
$ nano /etc/apt/sources.list # delete the cd repository if still
    ↪ there (seems like a debian 11 problem)
$ apt-get update
$ apt-get upgrade
$ apt-get install sudo vim git make gcc flex bison
    ↪ libncurses-dev libssl-dev libelf-dev bc ndctl
    ↪ original-awk
$ /sbin/adduser <non-sudo-user> sudo
$ vim /etc/ssh/sshd_config # add the line "PermitRootLogin yes"
$ systemctl restart sshd
```

Συνοπτικά, εγκαθιστούμε ότι πακέτο είναι απαραίτητο για την μεταγλώττιση του πυρήνα Linux και την εκτέλεση του πηγαίου κώδικα της παρούσας εργασίας, και ρυθμίζουμε το SSH ώστε να επιτρέπει την σύνδεση μέσω λογαριασμού διαχειριστή. Το τελευταίο αποτελεί δυνητικά αρκετά κακή πρακτική ασφαλείας, αλλά διευκολύνει σημαντικά την διαδικασία ανάπτυξης του πυρήνα μέσω της εικονικής μηχανής, με τον βασικότερο λόγο να αποτελεί η αποφυγή χρήσης της διεπαφής sudo για τροποποίηση και διάγνωση του συστήματος όταν θέλουμε να δοκιμάσουμε τις επιθυμητές αλλαγές στον πυρήνα. Συνδεόμαστε πλέον στην εικονική μηχανή με τον ακόλουθο τρόπο:

```
$ TERM=xterm ssh -p 2222 <root or user>@127.0.0.1
```

Ύστερα, διαμορφώνουμε τις εικονικές μη πτητικές μνήμες δημιουργώντας τα αντίστοιχα namespaces σε κατάσταση λειτουργίας fsdax:

```
$ ndctl create-namespace -f -e namespace0.0 --mode=fsdax
$ ndctl create-namespace -f -e namespace1.0 --mode=fsdax
```

Σε αυτό το σημείο η εικονική μηχανή είναι έτοιμη για τους σκοπούς ανάπτυξης του πυρήνα. Όπως αναφέρθηκε, προσομοιώνει επαρκώς ένα NUMA σύστημα με προσαρτημένες εικονικές μη πτητικές μνήμες, το οποίο είναι ικανοποιητικό για το kernel development. Ωστόσο, η ανάπτυξη του userspace component συναντάει περιορισμούς στο περιβάλλον της εικονικής μηχανής, καθώς δεν μπορούμε μέσω αυτού να κάνουμε δοκιμές και να αναπτύξουμε επί της δυνατότητας π.χ. για ανάγνωση των hardware μετρητών των συσκευών που είναι εφικτή μέσω του IPMCTL API της Intel. Θα περιοριστούμε λοιπόν στην ανάγνωση τιμών από το procfs και την προαναφερθείσα επέκταση υποχρεωτικά θα την εξετάσουμε σε μηχάνημα που διαθέτει πραγματικές μη πτητικές μνήμες.

3.4.3 Μεταγλώττιση του πυρήνα και εγκατάσταση

Σημειώνεται ότι η τελευταία έκδοση του GCC με την οποία λαμβάνουμε επιτυχή μεταγλώττιση είναι η ενδέκατη. Από την έκδοση 12 και μετά παρουσιάζεται ασυμβατότητα με την υλοποίηση της συνάρτησης `xrealloc` για την επιλεγμένη έκδοση του πυρήνα. Η μεταγλώττιση και εγκατάσταση του πυρήνα που αναπτύχθηκε για τους σκοπούς της παρούσας διπλωματικής μπορεί να γίνει ως εξής:

```
$ git clone https://github.com/PhTof/Diploma.git
$ cd Diploma/linux-5.13
$ cp ../various/minconfig_5.13 .config
$ make oldconfig
$ make -j4 && ./aftermake.sh && reboot
```

Έχουμε να παρατηρήσουμε δύο πράγματα εδώ, το πρώτο είναι το έτοιμο configuration `minconfig_5.13`, και το δεύτερο είναι το script `aftermake.sh`, τα οποία αναλύουμε ακολούθως.

Ελάχιστο Configuration

Διατίθεται με τον πηγαίο κώδικα της εργασίας αυτής ένα όσο το δυνατόν περιορισμένο kernel configuration για την εικονική μηχανή, αλλά πλήρες για την ενεργοποίηση της λειτουργίας των μη πτητικών μνημών και της λειτουργίας DAX. Οι βασικοί κανόνες μέσω των οποίων αυτό διαμορφώθηκε από το προεπιλεγμένο configuration, αποτελούν 1) η αφαίρεση driver που αφορούν πολλές δικτυακές και άλλες συσκευές (εξωτερικές συσκευές USB, κάρτες γραφικών κλπ.), που με μεγάλη βεβαιότητα δεν εξυπηρετούν την εικονική μηχανή μας, και

2) η προσθήκη όσων παραμέτρων του configuration είχαν αναγνωριστικό που περιλαμβάνει τους όρους "DAX," "PMEM" ή "NVDIMM."

Στην συνέχεια παρατίθεται ο πίνακας 3.3 που καταγράφει αναλυτικότερα τις σημαντικότερες επιλογές του προεπιλεγμένου configuration. Οι λόγοι που αφαιρέθηκαν αρκετοί αχρείαστοι driver άπτονται κυρίως στην ταχύτητα της εκ νέου μεταγλώττισης του πυρήνα (κάτι το οποίο θα κάνουμε συχνά για δοκιμή πιθανών αλλαγών που έχουμε πραγματοποιήσει), η οποία ταχύτητα βλάπτεται σημαντικά όταν επιβάλλεται η επιπλέον μεταγλώττιση πολλαπλών module που ούτως ή άλλως δεν χρειαζόμαστε.

Αναγνωριστικό του πυρήνα
[CONFIG_LOCALVERSION] General Setup > Local version - append to kernel release : Προσθήκη αναγνωριστικού για τον πυρήνα.

Υποστήριξη των μη πτητικών μνημών ως χώρο αποθήκευσης
[CONFIG_ACPI_NFIT] Power management and ACPI options > ACPI (Advanced Configuration and Power Interface) Support > ACPI NVDIMM Firmware Interface Table (NFIT) : Αναγνώριση μη πτητικών συσκευών κατά την εκκίνηση.
[FS_DAX] File systems > Direct Access (DAX) support : Υποστήριξη λειτουργίας άμεσης πρόσβασης (DAX).
[MEMORY_HOTPLUG] Memory Management options > Allow for memory hot-add : Εξάρτηση του [ZONE_DEVICE].
[MEMORY_HOTREMOVE] Memory Management options > Allow for memory hot remove : Εξάρτηση του [ZONE_DEVICE].
[ZONE_DEVICE] Memory Management options > Device memory (pmem, HMM, etc...) hotplug support : Παρέχει ικανότητα δημιουργίας namespaces με λειτουργία fsdax.
[LIBNVDIMM] Device Drivers > NVDIMM (Non-Volatile Memory Device) Support : Γενικότερη υποστήριξη μη πτητικών συσκευών, επιλέγεται αυτόματα από το [CONFIG_ACPI_NFIT].

Ρύθμιση σχετικά με την τοπολογία NUMA
[ACPI_HMAT] Power management and ACPI options > ACPI (Advanced Configuration and Power Interface) Support > NUMA support : Αναγνώριση στοιχείων τοπολογίας NUMA κατά την εκκίνηση.

Άλλες ρυθμίσεις σχετικές με τις μη πτητικές μνήμες

[X86_PMEM_LEGACY] Processor type and features > Support non-standard NVDIMMs and ADR protected memory : Υποστήριξη μη standard NVDIMMS.

[TRANSPARENT_HUGEPAGE] Memory Management options > Transparent Hugepage Support : Εξάρτηση του [CONFIG_DEV_DAX].

[CONFIG_DEV_DAX] Device Drivers > DAX: direct access to differentiated memory > Device DAX: direct access mapping device : Διαμόρφωση namespace σε devdax mode.

[CONFIG_DEV_DAX_KMEM] Device Drivers > DAX: direct access to differentiated memory > KMEM DAX: volatile-use of persistent memory : Χρήση των μη πτητικών μνημών ως RAM.

[CONFIG_DEV_DAX_PMEM] Device Drivers > DAX: direct access to differentiated memory > Device DAX: direct access mapping device : Raw access στην μη πτητική μνήμη.

[VIRTIO_PMEM] Device Drivers > Virtio drivers > Support for virtio pmem driver : Υποστήριξη για virtio-pmem συσκευές.

Ρυθμίσεις για δικτύωση

[CONFIG_ETHERNET] Device Drivers > Network device support > Ethernet driver support : Απενεργοποίηση όλων εκτός του Intel devices > Intel(R) PRO/100+ support [CONFIG_E1000]

[CONFIG_ACPI_PCI_SLOT] Power management and ACPI options > ACPI (Advanced Configuration and Power Interface) Support > PCI slot detection driver : Διαφορετικά έχουμε διεπαφή enp0s3 αντί για ens3 για την οποία έχει γίνει ρύθμιση από την εγκατάσταση της διανομής (PCI device εναντίον hotplug device). Αν προκύψει άλλη διεπαφή, χρειάζεται να εκτελέσουμε την εντολή `dhclient -d <interface>`.

Εναλλακτικές ρυθμίσεις για δικτύωση

[CONFIG_VIRTIO_CONSOLE] Device Drivers > Character devices > Virtio console : Εξάρτηση του [VIRTIO].

[VIRTIO_NET] Device Drivers > Network device support > Virtio network driver : Υποστήριξη VIRTIO διεπαφής δικτύου. Απαιτεί άλλες παραμέτρους στο QEMU, αλλά αποτελεί καλύτερη διεπαφή.

Ρυθμίσεις για ελαχιστοποίηση του configuration
[DRM_I915] Device Drivers > Graphics support : Δεν χρειαζόμαστε υποστήριξη γραφικών εφόσον θα συνδεόμαστε μέσω SSH.
[CONFIG_SURFACE_PLATFORMS] Device Drivers > Microsoft Surface Platform-Specific Device Drivers : Δεν χρειαζόμαστε vendor specific οδηγούς.
[CONFIG_MACINTOSH_DRIVERS] Device Drivers > Macintosh device drivers : Δεν χρειαζόμαστε vendor specific οδηγούς.
[CONFIG_SOUND] Device Drivers > Sound card support : Δεν χρειαζόμαστε υποστήριξη ήχου.
[CONFIG_USB_SUPPORT] Device Drivers > USB support : Δεν χρειαζόμαστε υποστήριξη USB, εκτός αν το USB passthrough διευκολύνει την μεταφορά αρχείων. Αυτό όμως μπορούμε να το κάνουμε σε κάποιο βαθμό μέσω της εντολής scp.
[CONFIG_WLAN] Device Drivers > Network device support > Wireless LAN : Έχουμε υποστήριξη δικτύου μέσω της εικονικής διεπαφής ethernet, οπότε δεν χρειαζόμαστε ασύρματη δικτύωση.
[CONFIG_WIRELESS] Networking support > Wireless : Περαιτέρω απενεργοποίηση της ασύρματης δικτύωσης.
[CONFIG_NETWORK_FILESYSTEMS] File systems > Network File Systems : Δεν είναι αναγκαία η χρήση δικτυακών συστημάτων αρχείων.
[CONFIG_MISC_FILESYSTEMS] File systems > Miscellaneous filesystems : Δεν χρειάστηκε κάποιο σύστημα αρχείων από αυτήν την κατηγορία.
Ρυθμίσεις για ικανότητα αποσφαλμάτωσης του πυρήνα
[CONFIG_DEBUG_INFO] Kernel hacking > Compile-time checks and compiler options : Υποστήριξη αποσφαλμάτωσης του πυρήνα.
[CONFIG_GDB_SCRIPTS] Kernel hacking > Compile-time checks and compiler options > Compile the kernel with debug info > Provide GDB scripts for kernel debugging : Καλύτερη υποστήριξη αποσφαλμάτωσης μέσω του GDB.

Πίνακας 3.3: Επιλογές για την διαμόρφωση του configuration του πυρήνα της εικονικής μηχανής

Οι επιλογές που αφορούν τις μη πτητικές μνήμες είναι σημαντικό να ελέγχονται πως είναι ενεργοποιημένες ακόμα και στο σύστημα αξιολόγησης,

καθώς για παράδειγμα δεν είναι σίγουρο ότι η προσαρμογή ενός παλιότερου configuration του συστήματος (εντολή `make oldconfig`) θα ρυθμίσει κατάλληλα το χαρακτηριστικό `FS_DAX`.

Επίσης, μεταξύ εγκαταστάσεων διαφορετικών εκδοχών του πυρήνα, χρειάζεται να βεβαιωθούμε ότι η ρύθμιση των namespace των συσκευών έχει παραμείνει ως έχει, ώστε διαφορετικά να την επαναφέρουμε. Συγκεκριμένα, αν εγκαταστήσουμε πυρήνα στον οποίο κάποια επιλογή που αφορά την λειτουργία DAX δεν έχει τεθεί σωστά ή δεν υφίσταται, τότε τα namespace που έχουμε ορίσει θα μεταβούν από την επιθυμητή κατάσταση fsdax στην raw. Ακόμα και ύστερα από την εγκατάσταση σωστά ρυθμισμένου πυρήνα, δεν θα αναιρεθεί απαραίτητα η μετάβαση της κατάστασης λειτουργίας. Είναι σημαντικό λοιπόν να πραγματοποιούμε έλεγχο μέσω της εντολής `ndctl list -N`, ή μέσω της εντολής `journalctl -b 0 -k` αφού δοκιμάσουμε να προσαρτήσουμε σύστημα αρχείων πάνω από συσκευή DAX με την επιλογή `-o dax`. Στην τελευταία περίπτωση θα εμφανιστεί σχετικό μήνυμα μη υποστήριξης της λειτουργίας DAX αν κάτι δεν έχει ρυθμιστεί σωστά.

Aftermake Script

Η διαδικασία μεταγλώττισης του πυρήνα δεν αποτελεί το μόνο σημείο χρονικής συμφόρησης, αφού κατά την εκκίνηση της εικονικής μηχανής, η ανάγνωση ενός μεγάλου initrd (initial ramdisk) αρχείου μπορεί να οδηγήσει σε μεγάλους χρόνους αναμονής. Αυτό μπορεί να αποδειχθεί χρονοβόρο σε περιπτώσεις που έχουμε ανάγκη να εξετάσουμε την λειτουργία κάποιας μικρής αλλαγής στον κώδικα του πυρήνα (πιθανώς εκτύπωση κάποιου διαγνωστικού μηνύματος μέσω της συνάρτησης `printk`). Για να περιορίσουμε την επίδραση αυτού του παράγοντα, φροντίσουμε πριν την παραγωγή του initrd και μετά την μεταγλώττιση του πυρήνα και των αντίστοιχων modules να αφαιρέσουμε από τα προαναφερθέντα οποιοδήποτε μη λειτουργικά αναγκαίο σύμβολο. Αυτό επιτυγχάνεται με την ακόλουθη εντολή:

```
$ find /lib/modules/<kernel_ver><local_ver>/ -name *.ko -exec
   ↪ strip --strip-unneeded {} +
```

Αυτό οδηγεί σε δραστική αλλαγή του μεγέθους του initrd αρχείου, και κατά συνέπεια σημαντική μείωση του χρόνου εκκίνησης της εικονικής μηχανής. Πιθανώς αυτό να μην αποτελεί καλή πρακτική, ή να υπάρχει διαθέσιμη κάποια επιλογή στο configuration του πυρήνα που να επιτρέπει την αφαίρεση των μη απαραίτητων συμβόλων μέσω της υποδομής μεταγλώττισης του. Για πρακτικούς όμως λόγους, έχει αποδειχτεί χρήσιμη.

Σαν μια πρώτη σημείωση αναφέρουμε ότι κατά την εκκίνηση της εικονικής

μηχανής με τον τροποποιημένο πυρήνα είναι πιθανό να λάβουμε ένα μήνυμα `"wrong EFI signature."` Αυτό ίσως να μπορεί να ερμηνευτεί από το γεγονός ότι το QEMU από μόνο του, χωρίς δηλαδή επιπρόσθετες παραμέτρους για την χρήση κάποιου διαθέσιμου UEFI firmware [22], χρησιμοποιεί BIOS ως υλικολογισμικό (firmware) εκκίνησης. Μπορούμε με κάποιες τροποποιήσεις στο configuration να αφαιρέσουμε την υποστήριξη περί UEFI από τον πυρήνα, αλλά δεν είναι αναγκαίο για τους σκοπούς αυτής της εργασίας.

Τέλος, σημειώνουμε ότι η μεταγλώττιση του τροποποιημένου πυρήνα που παρέχεται με τον πηγαίο κώδικα αυτής της εργασίας μέσω του προεπιλεγμένου configuration της έκδοσης πυρήνα 5.13 είναι πιθανό να οδηγήσει σε σφάλματα της μεταγλώττισης. Αυτό οφείλεται στην έλλειψη εναλλακτικών ορισμών των νεοεισαχθέντων συναρτήσεων/συμβόλων σε δομές `#if[n]def … #else … #endif`. Δεν έχει νόημα η μεταγλώττιση του πυρήνα αυτού αν δεν διατίθεται υποστήριξη για μη πτητικές συσκευές μέσω του configuration, για αυτό ο κώδικας έχει αφεθεί ως έχει. Η αναφορά στο γεγονός αυτό πραγματοποιείται για κάλυψη της περίπτωσης στην οποία ο αναγνώστης πιθανώς να αποπειραθεί να μεταγλωττίσει τον πυρήνα χωρίς να δώσει πρώτα ένα συμβατό configuration.

3.5 Tracing στον πυρήνα

Για οποιαδήποτε τροποποίηση θελήσουμε να πραγματοποιήσουμε στον κώδικα του πυρήνα, χρειάζεται πρώτα να εντοπίσουμε τις σχετικές συναρτήσεις του που υλοποιούν την λειτουργία την οποία θέλουμε να μεταβάλουμε. Για παράδειγμα, όπως θα δούμε στην ενότητα *Υλοποίηση*, για να τροποποιήσουμε κατάλληλα το σύστημα αρχείων ext4, θέλουμε πρώτα να εντοπίσουμε τις συναρτήσεις που σχετίζονται με την δέσμευση (μέτα)δεδομένων. Αν και το μονοπάτι εκτέλεσης σε κάθε περίπτωση μπορεί να είναι αρκετά σύνθετο και να περιλαμβάνει πολλές επιμέρους κλήσεις βοηθητικών συναρτήσεων, αρκούμαστε προς το παρόν στον ποιοτικό προσδιορισμό των υποψήφιων συναρτήσεων.

Επιπροσθέτως, μπορεί να υπάρχουν πολλές υποπεριπτώσεις που επηρεάζουν σε μεγάλο βαθμό το ποιες συναρτήσεις εκτελούνται και πως. Επανερχόμενοι στο παράδειγμα, μπορεί στο ext4 να υπάρχουν αρχεία που έχουν ρυθμιστεί ώστε να μην χρησιμοποιούν extent tree για δεικτοδότηση των block δεδομένων τους (απενεργοποίηση του flag `EXT4_INODE_EXTENT`). Ωστόσο εμείς θα περιοριστούμε στην πιο συνήθη περίπτωση, δηλαδή ό,τι μπορούμε να παρατηρήσουμε μέσω καταγραφής (tracing) των κληθέντων συναρτήσεων μια σειράς απλών πειραματισμών δημιουργίας καινούργιων αρχείων (δηλαδή inodes) και εγγραφής σε αυτά μέσω του FIO ή του Filebench.

3.5.1 Καταγραφή με το ftrace

Την καταγραφή των κληθέντων συναρτήσεων στον πυρήνα την επιτυγχάνουμε μέσω της ακόλουθης εντολής του `perf` (που εν προκειμένω λειτουργεί σαν ένας απλός wrapper του kernel ftrace [26]):

```
$ perf ftrace -a --inherit -T "<function_name>"
    ↪ <experiment_script>
```

Το σημαντικό είναι ότι η τιμή του πεδίου `<function_name>` μπορεί να είναι ακόμα και απλό regex. Συγκεκριμένα επιτρέπεται η χρήση του τελεστή αστερίσκος (*), και μάλιστα περισσότερες από μια φορές. Επομένως, αν θέλουμε για παράδειγμα να καταγράψουμε μέσω του perf τις συναρτήσεις που αφορούν το multiblock allocation, θέτουμε `<function_name>` = `ext4_mb_*`, ενώ αν θέλουμε απλά να καταγράψουμε κάθε σχετική κλήση με το ext4, μπορούμε να θέσουμε `<function_name>` = `*ext4_*`.

Σημειώνεται ότι ενδέχεται να χρειαστεί να ρυθμίσουμε και να μεταγλωττίσουμε κατάλληλα τον πυρήνα για να είναι εφικτό να χρησιμοποιηθεί το ftrace, ή ακόμα και για να επιτρέψουμε την καταγραφή σε συγκεκριμένα μέρη του κώδικα του πυρήνα.

Αν θέλουμε μέσω του `perf` να κάνουμε καταγραφή της αλληλουχίας κλήσεων που οδηγούν σε μια συνάρτηση που μας ενδιαφέρει, μπορούμε να επιστρατεύσουμε την ακόλουθη λογική:

```
#!/bin/bash

perf=$(echo $HOME/linux-5.13/tools/perf/perf)
func=submit_bio
executable=./simple

prepare_state() {
  ./simple # Be sure the file exists
}

restore_state () {
  : # Nothing to be done here
}

get_caller () {
       func=$1
       # tail -1: keep only the last call of the wanted function
       # awk '{print $NF}' | cut -c3- : output the caller
       $perf ftrace -T "$func" $executable | tail -1 | \
              awk '{print $NF}' | cut -c3-
}

prepare_state
```

```
while : ; do
        # Print the current function
        echo $func
        restore_state
        func=$(get_caller $func)
        if [[ -z "$func" || "$func" == "do_syscall_64" ]]; then
                break
        fi
done
```

Το εκτελέσιμο `./simple` στο παράδειγμα κάνει απλή εγγραφή σε ένα αρχείο. Θα μπορούσαμε να πετύχουμε κάτι παρόμοιο μέσω της εντολής `bash -c "echo ... > file"`. Αυτό ωστόσο πρέπει να αποφευχθεί ως πρακτική, γιατί ο τελεστής ανακατεύθυνσης έχει ιδιότροπη συμπεριφορά. Θα επεκταθούμε περισσότερο πάνω σε αυτή στην ενότητα *Αξιολόγηση Ορθότητας* του κεφαλαίου *Αξιολόγηση*. Επιπλέον, είναι επιθυμητό να περιορίσουμε όσο το δυνατόν την λειτουργία του παρεχόμενου εκτελέσιμου στην απολύτως επιθυμητή για να περιορίσουμε την πολυπλοκότητα της καταγραφής. Συγκεκριμένα, η εγγραφή του αρχείου μέσω του bash θα πραγματοποιούσε πολλές επιπλέον λειτουργίες που σχετίζονται με την υλοποίηση του ίδιου του bash παρά με την διαδικασία εγγραφής του αρχείου. Οι συναρτήσεις `prepare_state` και `restore_state` αποσκοπούν στην διαμόρφωση συνθηκών για να προκύψει η ίδια αλληλουχία κλήσεων κάθε φορά.

Σημειώνεται ότι κατά την μεταγλώττιση του πυρήνα πολλές συναρτήσεις που καλούνται από μόνο ένα σημείο τείνουν να ενσωματωθούν ("γίνονται inline") στην μοναδική καλούσα συνάρτηση για λόγους βελτιστοποίησης. Με αυτόν τον τρόπο όμως η συνάρτηση δεν θα τις συμπεριληφθεί στην λίστα συναρτήσεων που μπορούν να καταγραφούν (`/sys/kernel/tracing/available_filter_functions`). Σε αυτήν την περίπτωση χρειάζεται να προσθέτουμε το σύμβολο noinline στην αρχή του ορισμού της συνάρτησης που θέλουμε να καταγραφεί και να μεταγλωττίσουμε εκ νέου τον πυρήνα. Αυτό είναι χρονοβόρο, και πιθανώς να εξυπηρετεί περισσότερο κάποια από τις επόμενες μεθόδους.

Η πιο απλή και στην πράξη χρησιμοποιημένη μέθοδος εντοπισμού των ενσωματωμένων (inlined) συναρτήσεων είναι να καταγράψουμε την ελλειπή αλληλουχία κλήσεων, και να βρούμε τα ενδιάμεσα βήματα που απουσιάζουν με ανάγνωση του κώδικα μέσω του διαδικτυακού εργαλείου Elixir Bootlin. Το εργαλείο αυτό έχει πραγματοποιήσει μια στατική ανάλυση των συμβόλων του πυρήνα, και μπορεί για το καθένα να δώσει πιθανά σημεία ορισμού και αναφορών στον κώδικα της έκδοσης που έχουμε επιλέξει. Επομένως, αρκεί με ανάγνωση του κώδικα να ακολουθήσουμε την αλληλουχία των καταγεγραμμένων συναρτήσεων, και σε σημεία που λείπουν ενδιάμεσες συναρτήσεις, ακολουθού-

με κληθέντες συναρτήσεις με μια μόνο αναφορά για να δούμε αν θα καταλήξουμε κάπως στην επόμενη συνάρτηση της αλληλουχίας. Αυτή η διαδικασία μπορεί να αποδειχθεί χρονοβόρα αν έχουμε πολλές ενδιάμεσες, ενσωματωμένες συναρτήσεις, αλλά στην μέση περίπτωση μπορούμε σχετικά γρήγορα να τις εντοπίσουμε.

3.5.2 Χρήση της συνάρτησης dump_stack()

Αν θέλουμε να λάβουμε αναλυτική και ακριβή αλληλουχία κλήσεων για μια συγκεκριμένη συνάρτηση, τοποθετούμε μια κλήση της συνάρτησης `dump_stack()` σε οποίο σημείο της επιθυμούμε. Αυτή η πρακτική απαιτεί εκ νέου μεταγλώττιση του πυρήνα, και αν τοποθετηθεί η `dump_stack()` σε συνάρτηση που καλείται συχνά, θα έχουμε σημαντική επιβράδυνση στην εκτέλεση του λειτουργικού λόγω του journaling.

3.5.3 Δημιουργία υποδομής αποσφαλμάτωσης του πυρήνα μέσω εικονικής μηχανής

Η βέλτιστη λύση για να μπορούμε να έχουμε πλήρη εικόνα του τρόπου λειτουργίας του πυρήνα είναι να πραγματοποιήσουμε αποσφαλμάτωση όπως θα κάναμε σε οποιοδήποτε απλό εκτελέσιμο. Προφανώς αυτό δεν είναι τόσο απλό, εφόσον ο τρόπος εκτέλεσης ενός λειτουργικού συστήματος είναι πολύ σύνθετος και δεν μπορεί αυτό να εκτελεί ένα πρόγραμμα που θα πραγματοποιεί έλεγχο της ροής λειτουργίας του ίδιου του λειτουργικού. Ευτυχώς υπάρχει υποδομή αποσφαλμάτωσης του κώδικα του πυρήνα μέσω του QEMU σε συνδυασμό με το GDB (GNU Debugger).

Πρώτο βήμα μας είναι να μεταγλωττίσουμε τον πυρήνα. Αυτό μπορούμε να το κάνουμε είτε απευθείας στον ξενιστή, είτε στην εικονική μηχανή. Είναι σημαντικό σε κάθε περίπτωση να φροντίσουμε να εγκατασταθούν τα παραγόμενα module στο σύστημα της εικονικής μηχανής (όπως θα προέκυπτε από την εντολή `make modules_install`), και να ενσωματώσουμε στον πυρήνα οτιδήποτε θέλουμε να ελέγξουμε μέσω της αποσφαλμάτωσης. Για παράδειγμα, σε αυτήν την διπλωματική φροντίζουμε στην εντολή `make menuconfig` να επιλέξουμε το σύστημα αρχείων ext4 να ενσωματωθεί στον πυρήνα (`<*>`) αντί να μεταγλωττιστεί ως module (`<M>`).

Υπάρχουν τρία απαραίτητα για την αποσφαλμάτωση αρχεία που παρατίθενται στην συνέχεια. Στην περίπτωση μεταγλώττισης του πυρήνα εντός της εικονικής μηχανής, χρειάζεται να τα αντιγράψουμε μετά την μεταγλώττιση στο σύστημα ξενιστή. Προτιμήθηκε η πρακτική αυτή καθώς διευκολύνει την παραγωγή των αρχείων. Όλες οι εντολές που θα αναφερθούν εκτελούνται στον κατάλογο που περιέχει τον πηγαίο κώδικα του πυρήνα.

- `initrd`: Το initial ramdisk είναι απαραίτητο ώστε σε συνδυασμό με το kernel image `vmlinuz` να μπορεί να εκκινηθεί το QEMU απευθείας, χωρίς παρεμβολή κάποιου bootloader. Παράγεται μέσω της εντολής `make install` στον κατάλογο `/boot/` ως αρχείο της μορφής `initrd.img-<kernel-ver><local-ver>`, όπου `<kernel-ver>` είναι η έκδοση του πυρήνα (εδώ 5.13.0) και `<local-ver>` είναι η ονομασία που δώσαμε στον πυρήνα μέσω της παραμέτρου `CONFIG_LOCALVERSION` του configuration. Πριν την εντολή `make install` πρέπει να έχει μεταγλωττιστεί ο πυρήνας (`make -j`nproc``) και να έχουν εγκατασταθεί τα module (`make modules_install`).

- `vmlinuz`: Το vmlinuz αποτελεί το συμπιεσμένο εκτελέσιμο του πυρήνα χωρίς σύμβολα αποσφαλμάτωσης. Είναι το αρχείο `<kernel-dir>/arch/x86/boot/bzImage` που προκύπτει από την μεταγλώττιση του πυρήνα, όπου `<kernel-dir>` ο κατάλογος του μεταγλωττισμένου πηγαίου κώδικα του πυρήνα. Ισοδύναμα, στην διανομή debian το αρχείο αυτό βρίσκεται στον κατάλογο `/boot` μετά την εγκατάσταση του πυρήνα, και μπορεί να έχει την μορφή `vmlinuz-<kernel-ver><local-ver>`.

- `vmlinux`: Το vmlinux είναι το εκτελέσιμο του πυρήνα με σύμβολα αποσφαλμάτωσης, εφόσον αυτά έχουν ζητηθεί μέσω του configuration (επιλογή `CONFIG_DEBUG_INFO`). Παράγεται με την εντολή `make vmlinux`, και το αρχείο `vmlinux` καταλήγει στον κατάλογο που περιέχει τον πηγαίο κώδικα του πυρήνα.

Τα πρώτα δύο αρχεία αξιοποιούνται ως παράμετροι του QEMU, και επιτρέπουν την ενεργοποίηση της επιλογής απομακρυσμένης αποσφαλμάτωσης μέσω GDB στον ξενιστή. Το τρίτο εξυπηρετεί ως παράμετρος του GDB ώστε να μπορεί να αξιοποιήσει τον πίνακα συμβόλων του. Συγκεντρωτικά, στην εικονική μηχανή εκτελούμε τις ακόλουθες εντολές:

```
$ cd <kernel-dir>
$ make -j`nproc`
$ make modules_install
$ make install
$ make vmlinux
```

Έχοντας κάνει αυτήν την προετοιμασία, μεταφέρουμε τα απαιτούμενα αρχεία στον ξενιστή με τις επόμενες εντολές, όπου `<QEMU-dir>` ο κατάλογος στον ξενιστή στον οποίο έχουμε τα script του QEMU. Ύστερα απενεργοποιούμε την εικονική μηχανή. Θεωρούμε ότι `<ver> = <kernel-ver><local-ver>`.

```
$ scp -P 2222 root@127.0.0.1:/boot/vmlinuz-<ver> <QEMU-dir>
$ scp -P 2222 root@127.0.0.1:/boot/initrd.img-<ver> <QEMU-dir>
$ scp -P 2222 root@127.0.0.1:<kernel-dir>/vmlinux <QEMU-dir>
```

Έχοντας συλλέξει τα απαραίτητα αρχεία, ανανεώνουμε το script του QEMU προσθέτοντας τις ακόλουθες παραμέτρους:

- `-kernel vmlinuz-<ver>`: Η εικόνα του πυρήνα που θα χρησιμοποιήσουμε για την εκκίνηση.
- `-initrd initrd.img-<ver>`: Το initial ramdisk που θα αξιοποιήσει η εικονική μηχανή για την εκκίνηση.
- `-append "root=/dev/sda1 nokaslr console=ttyS0"`: Προσδιορίζουμε για την διαδικασία της εκκίνησης που βρίσκεται η βάση (root) του συστήματος αρχείων της εγκατάστασης Linux στην εικονική μηχανή (σε αυτήν την περίπτωση `/dev/sda`). Απενεργοποιούμε το kernel address space layout randomization (`nokaslr`) ώστε να υπάρχει αντιστοιχία μεταξύ των διευθύνσεων του vmlinux και αυτών στην εικονική μηχανή. Τέλος, θέλουμε για ευκολία η έξοδος του QEMU να εκτυπώνεται στο τερματικό από το οποίο εκτελέστηκε το script, αντί για ξεχωριστό παράθυρο. Αυτό μας επιτρέπει η παράμετρος `console=ttyS0` σε συνδυασμό με την επιλογή `-nographic` του QEMU.
- `-s -S`: Η επιλογή `-s` επιτρέπει την αποσφαλμάτωση του πυρήνα μέσω του QEMU, ενώ η επιλογή `-S` καθυστερεί την εκκίνηση της εικονικής μηχανής μέχρι να συνδεθεί με αυτήν το GDB.

Εκκινούμε την εικονική μηχανή, και προετοιμάζουμε το GDB στον host ως εξής:

```
$ gdb <QEMU-dir>/vmlinux
(gdb) target remote :1234
```

Σε αυτό το σημείο κάνουμε όποια προετοιμασία θέλουμε (π.χ. ορισμός breakpoint) και εκκινούμε την εικονική μηχανή δίνοντας την εντολή `c` στο GDB. Μπορούμε να διακόψουμε την εκτέλεση της εικονικής μηχανής οποιαδήποτε στιγμή πατώντας `Ctrl + C` στην διεπαφή του GDB. Από εδώ και στο εξής διαθέτουμε όλες τις ικανότητες που παρέχει το εργαλείο αποσφαλμάτωσης.

Αρκετά χρήσιμες εντολές είναι οι επόμενες: `b <function name>` για να ορίσουμε ένα breakpoint, `backtrace` για να εκτυπώσουμε την αλληλουχία κλήσεων συναρτήσεων που οδήγησε στην συνάρτηση στην οποία διακόπηκε η εκτέλεση, και `print <variable name>` για να εκτυπώσουμε το περιεχόμενο

μιας μεταβλητής που μας ενδιαφέρει (μπορούμε να επιλέξουμε από τις συναρτήσεις της αλληλουχίας μέσω της εντολής `f <frame number>`).

Αυτή η μέθοδος είναι πολύ ισχυρή, και έχει διπλό ρόλο αφού εξυπηρετεί και στην καταγραφή της αλληλουχίας συναρτήσεων που μας οδηγεί στο σημείο που μας ενδιαφέρει, και στην αποσφαλμάτωση του τροποποιημένου πυρήνα. Όμως, ενέχει μια χρονοβόρα, σύνθετη διαδικασία μεταφοράς και αξιοποίησης των απαραίτητων αρχείων, οπότε τείνουμε να την επιστρατεύουμε όταν οι υπόλοιπες μέθοδοι δεν είναι καρποφόρες.

Τέλος, ιδιαίτερα χρήσιμο στην ταχύτητα ανάγνωσης του κώδικα του πυρήνα αναδεικνύεται το εργαλείο `grep`. Έχοντας τον κώδικα του πυρήνα διαθέσιμο τοπικά, η ακόλουθη εντολή μπορεί να μας δώσει πολύτιμη πληροφορία για την εύρεση κάποιας συνάρτησης, τον ορισμό κάποιου struct κλπ:

```
$ grep -Irn "<pattern>" <kernel_dir>
```

Στην συνέχεια της παρούσας εργασίας, όποτε περιγράφουμε τα αποτελέσματα της διερεύνησης του πηγαίου κώδικα του πυρήνα θα αποφύγουμε να αναφέρουμε ρητά την μέθοδο εκ των παραπάνω που χρησιμοποιήθηκε.

Κεφάλαιο 4

Πειράματα

Σε αυτήν την ενότητα θα παρουσιάσουμε τα αποτελέσματα και τα πορίσματα από μια σειρά πειραμάτων για την μελέτη της συμπεριφοράς των μη πτητικών μνημών υπό συνθήκες εκτέλεσης εργασιών I/O από πολλαπλά νήματα. Το βασικό εργαλείο που θα βοηθήσει τον πειραματισμό σε αυτήν την ενότητα είναι το FIO. Εστιάζουμε με τα πειράματα κυρίως στο πως εκφυλίζεται η επίδοση των μη πτητικών μνημών παρουσίας νημάτων που επιτελούν απομακρυσμένες προσβάσεις, δηλαδή στέλνουν αιτήματα σε μη πτητικές μνήμες μη τοπικού NUMA κόμβου. Από τις μετρήσεις συμπεραίνουμε ότι υπάρχει μια βασική ευπάθεια των μη πτητικών μνημών, που σχετίζεται με την μείωση της ρυθμαπόδοσης σε συνθήκες εκτέλεσης συγκεκριμένης αλληλουχίας προσβάσεων.

Στην επόμενη ενότητα θα δούμε πως η ευπάθεια που αναφέραμε συνδράμει στον σχηματισμό της αφόρμησης της εργασίας. Επιπροσθέτως, περισσότερα στοιχεία σχετικά με το πως μπορούμε να διερευνήσουμε περαιτέρω τα φαινόμενα που παρατηρούμε σε αυτήν την ενότητα αναφέρονται στο κεφάλαιο *Παράρτημα*.

4.1 Χαρακτηριστικά του συστήματος αξιολόγησης

Στην συνέχεια παρουσιάζουμε κάποια τεχνικά χαρακτηριστικά του υπολογιστικού συστήματος που αποτέλεσε την υποδομή αξιολόγησης. Ανακτούμε τις σχετικές πληροφορίες του πίνακα μέσω των ακόλουθων εντολών:

```
$ lscpu
$ sudo ipmctl show -dimm
$ sudo ipmctl show -a -dimm 0x0000
$ sudo dmidecode -t 17
```

Οι σημαντικότερες παράμετροι του είναι η ύπαρξη δύο NUMA κόμβων και η εγκατάσταση τριών μη πτητικών μνημών ανά κόμβο. Έτσι, ανά κόμβο αναμένουμε να έχουμε αθροιστικό bandwidth ίσο με 19.8 GB/s για αναγνώσεις και 6.9 GB/s για εγγραφές.

Γενικά χαρακτηριστικά	
Έκδοση πυρήνα	5.13.0
Διανομή GNU/Linux	Ubuntu 18.04.6 LTS
Επεξεργαστής	
Μοντέλο	Intel(R) Xeon(R) Gold 5218T CPU @ 2.10GHz
Αριθμός επεξεργαστικών πυρήνων	32
Νήματα ανά επεξεργαστικό πυρήνα	2
Αριθμός NUMA κόμβων	2
Εύρος συχνότητας λειτουργίας	1.0 GHz – 3.8 Ghz
Προσωρινή μνήμη (Cache)	L1d: 32K, L1i: 32K L2: 1024K L3: 22528K
Μη πτητικές μνήμες	
Μοντέλο	NMA1XXD128GPS
Χωρητικότητα ανά μονάδα	126.4 GiB
Αριθμός μονάδων ανά NUMA κόμβο	3
Έκδοση Firmware	01.02.00.5355 ή 01.02.00.5446
Μνήμες RAM	
Μοντέλο	36ASF4G72PZ-2G9E2
Χωρητικότητα ανά μονάδα	32 GiB
Αριθμός μονάδων ανά NUMA κόμβο	3
Χρονισμός	2666 MT/s

Πίνακας 4.1: Χαρακτηριστικά του συστήματος αξιολόγησης

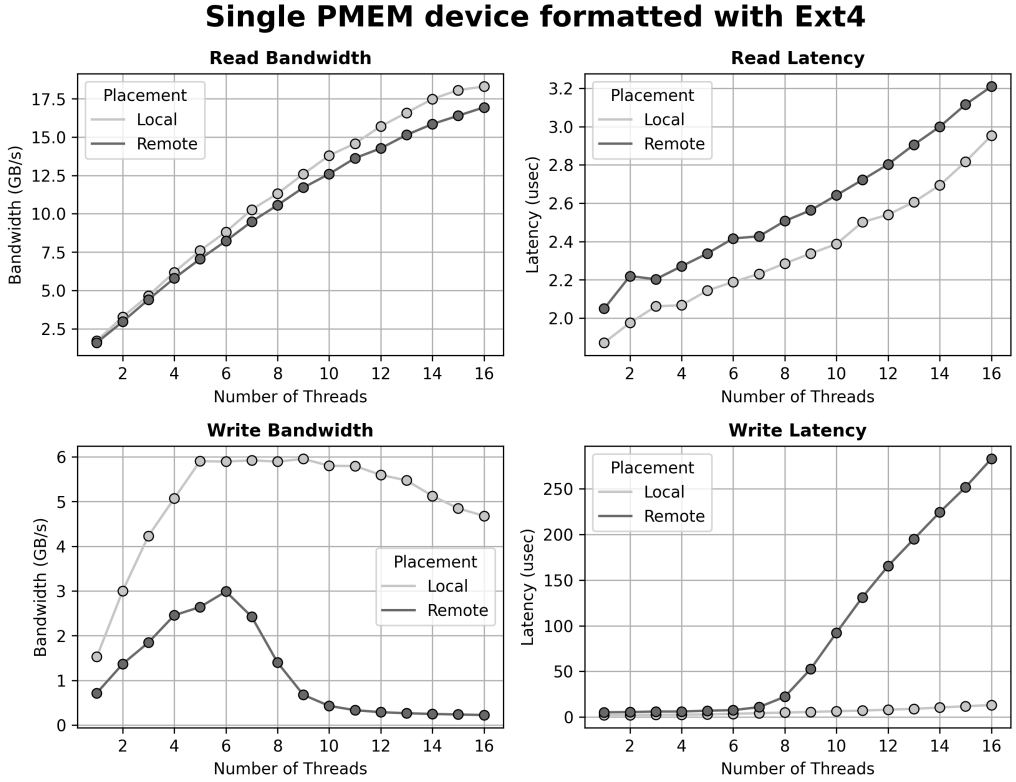
Εστιάζουμε στην υποστήριξη υπερνημάτωσης (hyperthreading), όπως φαίνεται από το γεγονός ότι έχουμε 2 νήματα ανά επεξεργαστικό πυρήνα, άρα και 64 λογικούς πυρήνες.

4.2 Πειράματα με το FIO

Σε αυτήν την ενότητα θα επιχειρήσουμε να εξετάσουμε την συμπεριφορά των μη πτητικών μνημών σε περιβάλλον αρχιτεκτονικής NUMA.

4.2.1 Θεμελιώδη χαρακτηριστικά των μνημών

Το πρώτο πείραμα που εκτελούμε αποσκοπεί στην εξέταση των θεμελιωδών χαρακτηριστικών των τοπικών και απομακρυσμένων προσβάσεων στις μη πτητικές μνήμες. Χρησιμοποιούμε ως σύστημα αρχείων το ext4 χωρίς καμία τροποποίηση, πάνω στην συσκευή `/dev/pmem0` που αναλογεί σε όλες τις μη πτητικές συσκευές του NUMA κόμβου 0.



Σχήμα 4.1: Θεμελιώδη χαρακτηριστικά των μη πτητικών μνημών (χρήση τελευταίου δείγματος)

Έχουμε δύο είδη τοποθέτησης των νημάτων: στην τοπική τοποθέτηση (local placment) όλα τα νήματα βρίσκονται στον κόμβο 0 που βρίσκονται οι μνήμες, ενώ στην απομακρυσμένη τοποθέτηση (remote placement) όλα τα νήματα βρίσκονται στον κόμβο 1. Κάθε νήμα πραγματοποιεί πρόσβαση σε δικό του αρχείο,

και οι προσβάσεις είναι είτε μόνο αναγνώσεις, είτε μόνο εγγραφές. Η προσπέλαση των αρχείων γίνεται σειριακά. Λαμβάνουμε ανά περίπτωση τρία δείγματα, και κρατάμε το πιο πρόσφατο από αυτά. Οι επιπλέον λεπτομέρειες του πειράματος διατίθενται στο αρχείο `experiments/scenarios/primitive.sh` του πηγαίου κώδικα της εργασίας.

Σημειώνεται ότι για να λάβουμε μικρότερο θόρυβο άρα και πιο ομαλές καμπύλες, χρησιμοποιήσαμε αρχεία μεγέθους τουλάχιστον ίσου με 1 GB, εν προκειμένω 4. Οι μετρικές που εξετάζουμε είναι η ραθμαπόδοση και ο λανθάνων χρόνος, ενώ η μεταβλητή του οριζόντιου άξονα είναι ο αριθμός νημάτων που εκτελούνται ταυτόχρονα.

Ως προς τις αναγνώσεις, βλέπουμε ότι η ρυθμαπόδοση δεν φαίνεται να απέχει πολύ μεταξύ των περιπτώσεων τοποθέτησης των αρχείων. Στο latency αναμενόμενα υπάρχει μια στοιχειώδης καθυστέρηση στην περίπτωση της απομακρυσμένης ανάγνωσης, που οφείλεται στο δίκτυο διασύνδεσης των επεξεργαστών.

Στις εγγραφές είναι πιο σύνθετα τα αποτελέσματα. Αρχικά παρατηρούμε ότι ακόμα και στην τοπική πρόσβαση, το bandwidth των μη πτητικών μνημών εξαντλείται πολύ γρήγορα, και μετά από ένα σημείο μειώνεται κιόλας. Στις απομακρυσμένες προσβάσεις η κατάσταση είναι αρκετά χειρότερη. Έχουμε στην καλύτερη περίπτωση υποδιπλάσια αξιοποίηση του bandwidth σε σχέση με τις τοπικές προσβάσεις, αλλά και πολύ απότομο εκφυλισμό της επίδοσης όσο αυξάνονται τα νήματα. Παρατηρούμε επίσης το καταγεγραμμένο latency να αυξάνεται ραγδαία.

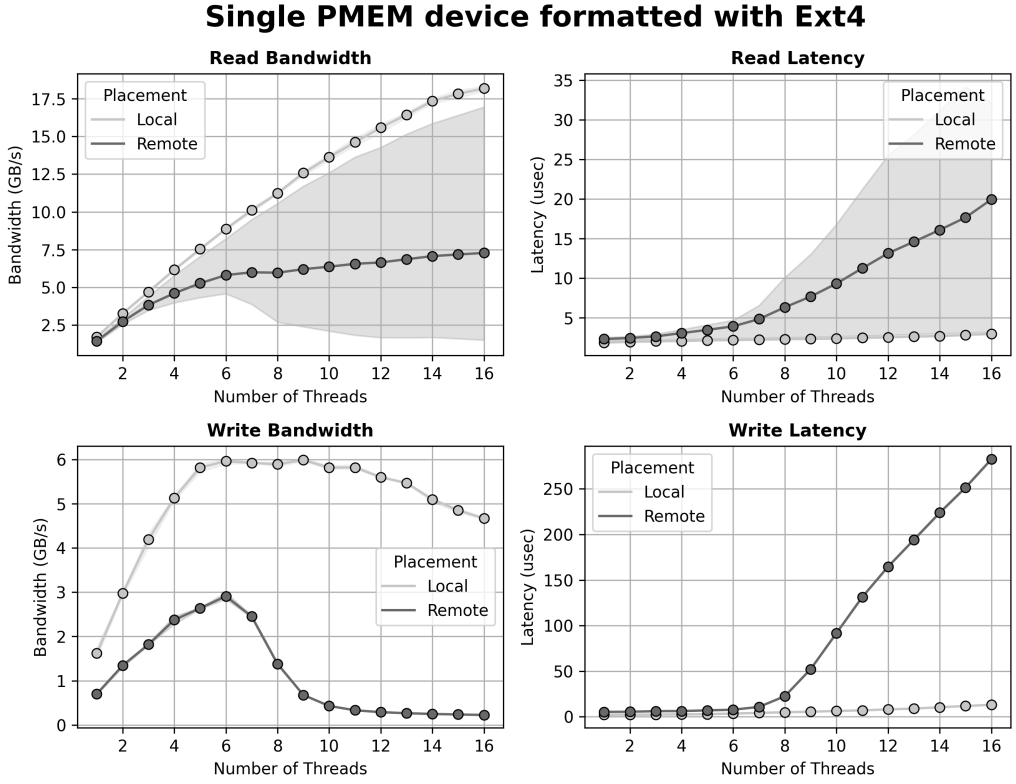
Το άμεσο συμπέρασμα που εξάγουμε από αυτό το διάγραμμα είναι ότι είναι σκόπιμη η αποφυγή των απομακρυσμένων εγγραφών για να διατηρήσουμε την επίδοση των συσκευών. Αυτό είναι κάτι βέβαια που έχει καταγραφεί προηγουμένως στην βιβλιογραφία [30].

4.2.2 Το φαινόμενο ανάγνωσης μετά από απομακρυσμένη εγγραφή

Αν κοιτάξουμε πιο προσεκτικά τα δείγματα που λάβαμε από την εκτέλεση του πειράματος για κάθε μέτρηση, παρατηρούμε ότι υπάρχει μια μη αμελητέα διακύμανση μεταξύ τους. Συγκεκριμένα, αν επιτρέψουμε να γίνει το διάγραμμα που προκύπτει από την μέση τιμή των δειγμάτων κάθε μέτρησης με σκιαγράφηση της διακύμανσης, λαμβάνουμε την εικόνα που φαίνεται στο σχήμα 4.2.

Στις εγγραφές δεν υπάρχει καμία διαφοροποίηση, αλλά οι απομακρυσμένες αναγνώσεις εμφανίζουν μεγάλη μεταβλητότητα και αισθητά χειρότερη επίδοση από αυτήν που περιμέναμε βάσει του σχήματος 4.1. Το πρώτο δείγμα κάθε μέτρησης, που αντιστοιχεί στο κάτω όριο της σκιαγραφημένης περιοχής για

την περίπτωση του bandwidth, μοιάζει να ακολουθεί την συμπεριφορά μιας απομακρυσμένης εγγραφής. Το δεύτερο δείγμα, όπως υποδεικνύει ο μέσος όρος, καταγράφει λίγο καλύτερη επίδοση, αλλά και πάλι αρκετά χειρότερη σε σχέση με το τι θα περιμέναμε. Η τρίτη μέτρηση, που αναλογεί στο άνω όριο της σκιαγραφημένης περιοχής, είναι αυτή που παρουσιάζει την αναμενόμενη επίδοση.



Σχήμα 4.2: Θεμελιώδη χαρακτηριστικά των μη πτητικών μνημών (χρήση όλων των δειγμάτων)

Μετά από σχετική διερεύνηση, έγινε αντιληπτό πως ο λόγος που παρατηρείται το φαινόμενο αυτό είναι το γεγονός ότι πριν από κάθε μέτρηση διαγράφονταν τα αρχεία του FIO ώστε να δημιουργηθούν καινούργια (ακολουθώντας την αρχή σύμφωνα με την οποία θέλουμε να έχουμε πάντα ίδια αρχική κατάσταση στα πειράματα). Η δημιουργία του αρχείου προϋποθέτει εγγραφή σε αυτό. Επομένως, το πρώτο δείγμα έχει την χειρότερη επίδοση και είναι αυτό που κάνει πρώτη φορά ανάγνωση μετά την εγγραφή του αντίστοιχου αρχείου. Το αξιοσημείωτο εδώ είναι ότι επηρεάζεται και το δεύτερο δείγμα, παρόλο που δεν ακολουθεί άμεσα την εγγραφή όπως το πρώτο. Μόνο με το τρίτο δείγμα λαμβάνουμε κάτι διαισθητικά αποδεκτό.

Αυτό το φαινόμενο θα το αναγνωρίζουμε στο εξής ως "Read After Remote Write." Σύμφωνα με αυτό, οι αναγνώσεις που ακολουθούν μια απομακρυσμένη εγγραφή έχουν σημαντικά μειωμένη επίδοση. Ο λόγος που συμβαίνει δεν ήταν εφικτό να διαπιστωθεί στα χρονικά πλαίσια της εργασίας, αλλά παρατίθεται η σχετική διερεύνηση που έγινε, στην ενότητα *Διερεύνηση του φαινομένου "Read After Remote Write"* του παραρτήματος.

Καταλαβαίνουμε λοιπόν ότι δεν είναι προβληματικές αποκλειστικά οι απομακρυσμένες εγγραφές, αλλά υπό συνθήκες όλες οι απομακρυσμένες προσβάσεις μπορεί να έχουν μειωμένη επίδοση. Μάλιστα, οι μικτές προσβάσεις, δηλαδή το να κάνουμε σε ένα αρχείο και αναγνώσεις και εγγραφές, δεν είναι καθόλου σπάνιο φαινόμενο. Για αυτό στην επόμενη υποενότητα εξετάζουμε τι γίνεται σε ένα τέτοιο σενάριο.

4.2.3 Συμπεριφορά των μικτών προσβάσεων

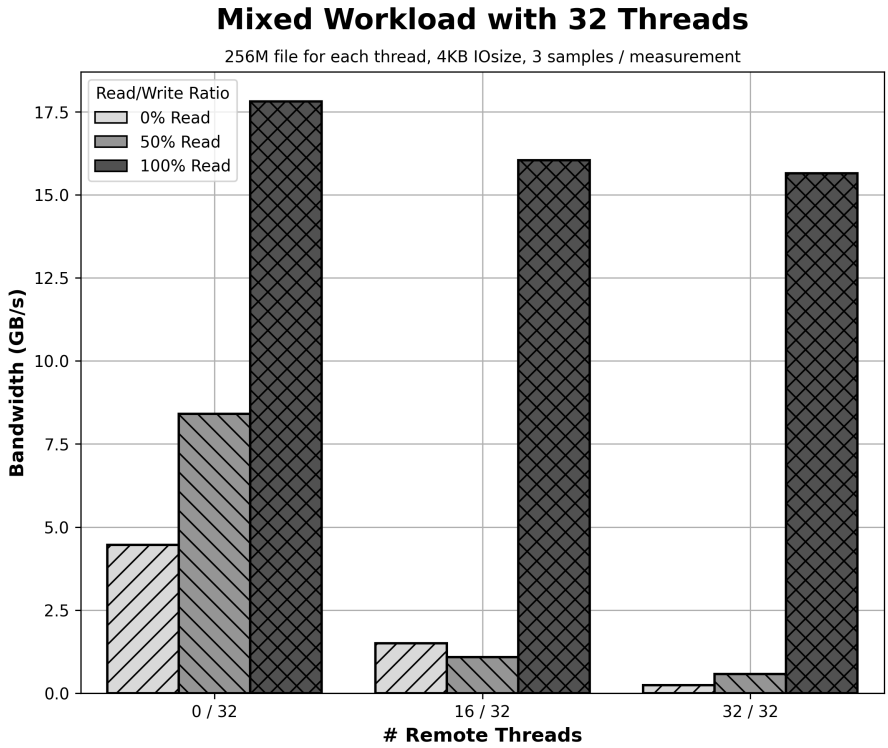
Στο δεύτερο πείραμα που εκτελούμε επιθυμούμε να εξετάσουμε τι γίνεται αν εκτός από καθαρές εγγραφές και καθαρές αναγνώσεις έχουμε μια περίπτωση στην οποία οι μισές προσβάσεις που κάνουμε στο αρχείο είναι αναγνώσεις, και οι άλλες μισές είναι εγγραφές. Αυτό ρυθμίζεται άμεσα μέσω της παραμέτρου rwmix του FIO.

Μέχρι τώρα η πολιτική της κατανομής υπήρξε όλα τα νήματα να είναι στον ίδιο κόμβο, άρα είχαμε τις περιπτώσεις local και remote placement. Θα θέλαμε να εξετάσουμε επίσης πόσο διαφορετική είναι η συμπεριφορά του πειράματος αν επιτρέψουμε μια ακόμα κατάσταση στην οποία κάποια νήματα βρίσκονται στον έναν κόμβο και τα υπόλοιπα στον άλλον. Υπενθυμίζεται ότι αξιοποιούμε τις μη πτητικές συσκευές μόνο του ενός κόμβου, άρα η μια κατηγορία νημάτων θα κάνει τοπικές προσβάσεις, ενώ η άλλη απομακρυσμένες, και μπορούν να συνυπάρχουν οι δύο αυτές κατηγορίες στην ίδια εκτέλεση. Θεωρούμε ότι έχουμε σταθερό αριθμό νημάτων, ίσο με 32, και μεταβάλλουμε την κατανομή τους μεταξύ των κόμβων. Για κάθε μέτρηση λαμβάνουμε τρία δείγματα, και κρατάμε το καλύτερο.

Λαμβάνουμε λοιπόν το αποτέλεσμα που φαίνεται στο σχήμα 4.3. Οι διαφορετικές αποχρώσεις αντιστοιχούν στο ποσοστό των προσβάσεων που είναι αναγνώσεις, ενώ ο οριζόντιος άξονας υποδεικνύει πόσα από τα 32 νήματα είναι απομακρυσμένα.

Παρατηρούμε τα εξής:

1. Στην περίπτωση που όλα τα νήματα είναι τοπικά, τα νούμερα είναι σχετικά αναμενόμενα. Η περίπτωση μικτών προσβάσεων δεν είναι ακριβώς ο γραμμικός συνδυασμός των καθαρών αναγνώσεων και εγγραφών, αλλά δεν είναι έξω από τα όρια του επιτρεπτού.



Σχήμα 4.3: Συμπεριφορά των μη πτητικών μνημών για μικτές προσβάσεις και κατανομές νημάτων

2. Στην περίπτωση που όλα τα νήματα είναι απομακρυσμένα, βλέπουμε ξανά την θεαματική διαφορά μεταξύ εγγραφών και αναγνώσεων, και μάλιστα οι μικτές προσβάσεις έχουν αθροιστικό bandwidth πάρα πολύ κοντινό στις εγγραφές. Αυτό από μόνο του αναδεικνύει πόσο επιβλαβές είναι το φαινόμενο "Read After Remote Write."

3. Στην περίπτωση της ισόποσης κατανομή νημάτων μεταξύ κόμβων, η εικόνα που λαμβάνουμε είναι πιο κοντά στην περίπτωση των καθαρά απομακρυσμένων προσβάσεων παρά σε αυτήν των καθαρά τοπικών προσβάσεων. Εκτός αυτού, εδώ όχι μόνο οι μικτές προσβάσεις είναι υπερβολικά επιβαρυμένες, αλλά είναι και ελαφρώς χειρότερες από τις καθαρές εγγραφές.

Καταλαβαίνουμε λοιπόν ότι λόγω του φαινομένου "Read After Remote Write" είναι ύψιστης σημασίας για την συνολική επίδοση του συστήματος να αποφεύγεται κατά το δυνατό οποιαδήποτε μορφή απομακρυσμένης πρόσβασης.

Κεφάλαιο 5

Κίνητρο της Εργασίας

Σε σχέση με άλλα μέσα αποθήκευσης, οι μη πτητικές μνήμες παρέχουν αξιοσημείωτο συγκριτικό πλεονέκτημα με μετρική την επίδοση τους. Παράλληλα όμως, η τεχνολογία τους δεν τους επιτρέπει την ίδια χωρητικότητα ανά συσκευή σε σύγκριση με πιο παραδοσιακά αποθηκευτικά μέσα, και ανά κόμβο μπορούμε να προσαρτήσουμε περιορισμένο αριθμό συσκευών. Το γεγονός αυτό, σε συνδυασμό με την ανάγκη να συμβαδίσουμε με την κλιμάκωση που επιδιώκει να επιτύχει η αρχιτεκτονική NUMA, μας οδηγεί στην θεώρηση μιας λογικής διεπαφής που θα ενοποιεί με αποδοτικό τρόπο το σύνολο των μη πτητικών συσκευών του συστήματος, ανεξαρτήτως NUMA κόμβου.

Η αναφερόμενη διεπαφή θα κάνει διαθέσιμη την συνδυασμένη χωρητικότητα των συσκευών κάτω από ένα ενιαίο επίπεδο αφαίρεσης, πιο συγκεκριμένα ένα ειδικά προσαρμοσμένο σύστημα αρχείων που βάσει της κατασκευής του θα αποφεύγει κατά το δυνατό τις παθογένειες που παρουσιάστηκαν στο κεφάλαιο *4*. Θα μπορούσαν να αναφερθούν διάφορες προτάσεις προς αυτήν την κατεύθυνση, ανάμεσα στις οποίες η μετακίνηση δεδομένων μεταξύ μνημών διαφορετικών κόμβων, ή ακόμα και η κατά συνθήκες αρωγή της page cache στην ελαχιστοποίηση των απομακρυσμένων προσβάσεων. Μάλιστα, η τελευταία ιδέα έχει διερευνηθεί ελάχιστα στην έως τώρα βιβλιογραφία.

Βασική σχεδιαστική αρχή: Για διατήρηση απλότητας και αποδοτικότητας στον σχεδιασμό, επιδιώκουμε να ευνοήσουμε την πιο απλή περίπτωση: θέλουμε να διασφαλίσουμε ότι οι δεσμεύσεις χώρου θα γίνονται σε υποκείμενες μνήμες που ανήκουν στον ίδιο NUMA κόμβο με αυτόν της αιτούσας διεργασίας. Ομολογουμένως, αυτή η ιδιότητα από μόνη της δεν αποδεικνύεται ιδιαίτερα χρήσιμη σε περιπτώσεις που αξιοποιούνται ήδη υπάρχοντα αρχεία ξένων κόμβων, χωρίς αυτά να δέχονται επεκτάσεις. Ωστόσο, βασιζόμαστε στην παραδοχή ότι μια διεργασία κατά την διάρκεια ζωής της δημιουργεί μέρος του συνόλου αρχείων πάνω στο οποίο εργάζεται ή συχνά μόνο επεκτείνει κάποια ήδη υπάρχοντα αρχεία του συνόλου. Αν η διεργασία δεν δρομολογείται υπερ-

βολικά συχνά σε διαφορετικούς κόμβους, είναι ξεκάθαρο ότι η ιδιότητα των τοπικών δεσμεύσεων δυνητικά θα αυξήσει και το ποσοστό των τοπικών προσβάσεων, επιφέροντας κέρδος στην επίδοση.

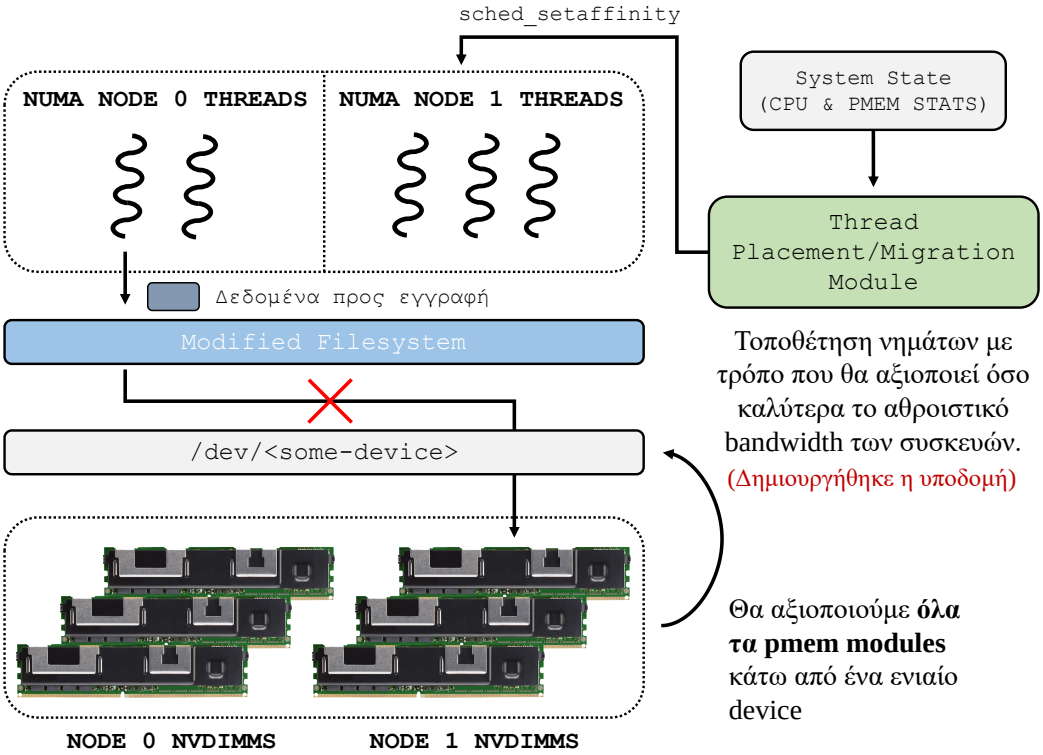
Επιλογή συστήματος αρχείων. Έχοντας διαμορφώσει τον βασικό μας στόχο, μεταβαίνουμε στην επιλογή κατάλληλου συστήματος αρχείων. Την στιγμή συγγραφής της εργασίας, διατίθενται 2 ερευνητικά συστήματα αρχείων που έχουν κατασκευαστεί με τις ιδιαιτερότητες των μη πτητικών μνημών ενσωματωμένες στις σχεδιαστικές επιλογές τους. Αυτά τα συστήματα αρχείων είναι ονομαστικά τα NOVA [29] και WineFS [18]. Ωστόσο, κανένα από τα δύο δεν διαθέτει ευρεία υιοθέτηση, και επιτρέπουν αποκλειστικά λειτουργία άμεσης πρόσβασης (DAX) στις συσκευές. Το γεγονός αυτό μπορεί να εισάγει δυσκολίες αν μελλοντικά επεκτείνουμε την εργασία αυτή για να συμπεριλαμβάνει κατά συνθήκη την page cache στο μονοπάτι δεδομένων μεταξύ μη πτητικών μνημών και διεργασιών.

Κατάλληλο υποψήφιο σύστημα αρχείων κρίνεται το ext4. Αν και υστερεί στο γεγονός ότι δεν είναι ειδικά σχεδιασμένο για λειτουργία με μη πτητικές μνήμες, είναι ευρέως διαδεδομένο και διαθέτει υποστήριξη ρύθμισης του χαρακτηριστικού DAX, μάλιστα ανά αρχείο. Συγκεκριμένα, από την έκδοση 5.13, η μόνη προϋπόθεση για την εναλλαγή τρόπου πρόσβασης στο αρχείο είναι να μην είναι ανοικτό από κάποια διεργασία. Σε προηγούμενες εκδόσεις υπήρχαν και άλλες προϋποθέσεις, που έκαναν πρακτικά ανέφικτη την δυναμική ρύθμιση του χαρακτηριστικού [8] [9]. Αυτό επιτρέπει πολύ ευκολότερα την πιθανή μελλοντική επέκταση της εργασίας που αναφέρθηκε προηγουμένως.

Σύστημα κατανομής νημάτων σε NUMA κόμβους: Αφού επιτευχθεί η υλοποίηση της ιδιότητας των τοπικών δεσμεύσεων στο ext4, είναι λογική απόρροια η επιθυμία ενός μηχανισμού που θα εξετάζει παραμέτρους της κατάστασης του συστήματος και θα επιλέγει κάθε φορά τον καλύτερο NUMA κόμβο για να τοποθετηθεί κάθε νήμα εκτέλεσης. Οι παράμετροι που αναφέρθηκαν θα μπορούσαν να είναι μεταξύ άλλων η πίεση στις μη πτητικές μνήμες (συνολικά byte από αιτήματα αναγνώσεων και εγγραφών σε αυτές), σε ποιον NUMA κόμβο πραγματοποιεί τις περισσότερες προσβάσεις μια διεργασία, το ποσοστό αξιοποίησης των επεξεργαστών κάθε κόμβου, ο διαμοιρασμός δεδομένων μεταξύ διεργασιών κλπ. Ένα παράδειγμα τέτοιου μηχανισμού είναι αυτό της δημοσίευσης [28].

Σε αυτήν την εργασία δημιουργήθηκε μια βασική υποδομή σε επίπεδο κώδικα, αλλά δεν θα παρουσιαστούν σχετικά πειραματικά αποτελέσματα, αφού το σύστημα είχε αρκετές ελλείψεις και λανθασμένες προσεγγίσεις στον σχεδιασμό ώστε να μην ήταν δυνατή η ολοκλήρωση του στα χρονικά πλαίσια της εργασίας. Ωστόσο, ο κώδικας είναι διαθέσιμος στον πηγαίο που συνοδεύει την παρούσα εργασία, ώστε να διευκολυνθεί η οποία πιθανή απόπειρα επέκτασης της.

Στην συνέχεια παρουσιάζεται σχηματικά ο σχεδιασμός που περιγράφτηκε:



Σχήμα 5.1: Η υποδομή που θέλουμε να δημιουργήσουμε στην παρούσα εργασία

Κεφάλαιο 6

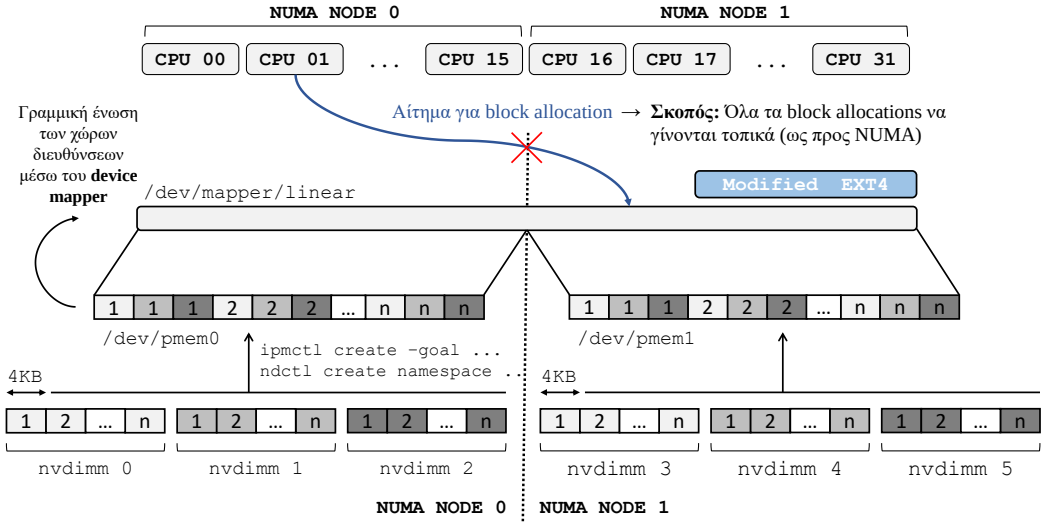
Υλοποίηση

Σε αυτήν την ενότητα θα παρουσιαστεί η διαδικασία ανάπτυξης του συστήματος που προτάθηκε στο κεφάλαιο 5. Θα δούμε, με την σειρά που αναφέρονται, τον τρόπο δημιουργίας της ενιαίας συσκευής που ενοποιεί όλες τις μη πτητικές συσκευές του συστήματος, τις επεκτάσεις στο ext4 για υποστήριξη NUMA aware δεσμεύσεων στην προαναφερθείσα συσκευή, την δημιουργία υποδομής στον πυρήνα για σωστή συλλογή στατιστικών I/O ανά NUMA κόμβο, και τέλος την προεργασία που έγινε προς την δημιουργία του userspace component.

6.1 Ενοποίηση των μη πτητικών συσκευών

Όπως αναφέραμε, θέλουμε μέσω του συστήματος που αναπτύσσουμε για τους σκοπούς της εργασίας να κάνουμε διαχείριση όλων των μη πτητικών συσκευών, ανεξαρτήτως NUMA κόμβου. Υπενθυμίζεται ότι μέσω του εργαλείου IPMCTL της Intel μπορούμε να λάβουμε ανά NUMA κόμβο τμηματική συνένωση των διαθέσιμων συσκευών σε επίπεδο υλικού (αποδιδόμενο καλύτερα με αγγλικούς όρους: "hardware interleaving across the requested modules"). Αυτό επιτυγχάνεται με αρκετά αποδοτικό τρόπο ώστε να μην υπάρχει λόγος να επιδιώξουμε να διαχειριστούμε τις συσκευές σε επίπεδο ενός NUMA κόμβου με διαφορετικό τρόπο. Επομένως, όταν μιλάμε για συνένωση συσκευών στο εξής, θα εννοούμε συνένωση των αρχείων συσκευών `/dev/pmem<X>` που προκύπτουν από το IPMCTL για κάθε NUMA κόμβο.

Θα μπορούσαμε προς την κατεύθυνση της ενοποίησης να παρέχουμε πολλαπλά αρχεία συσκευών ως παραμέτρους κατά την δημιουργία του επιλεγμένου συστήματος αρχείων, και εσωτερικά αυτού να εμπεριέχεται η λογική χρήσης της κατάλληλης για κάθε πρόσβαση συσκευής. Παράδειγμα τέτοιας περίπτωσης υλοποίησης φαίνεται στην επέκταση του συστήματος αρχείων NOVA



Σχήμα 6.1: Σχηματική αναπαράσταση του τρόπου ενοποίησης των μη πτητικών συσκευών

που πραγματοποιήθηκε για τους σκοπούς σύγκρισης με το σύστημα αρχείων WineFS [18] [21]. Ωστόσο, αυτό θα προσέθετε σημαντική πολυπλοκότητα στις τροποποιήσεις που πρόκειται να αναπτύξουμε σε επίπεδο συστήματος αρχείων. Επιπλέον, θα οδηγούσε σε μεγάλη επικάλυψη λογικής με εκτενώς ελεγμένους και άμεσα διαθέσιμους οδηγούς RAID σε επίπεδο λογισμικού.

Βάσει των όσων αναφέρθηκαν, καταλήγουμε στην προτίμηση δημιουργίας ενός αρχείου συσκευής που αναλαμβάνει την ενοποίηση των συσκευών μέσω κάποιου διαθέσιμου οδηγού του πυρήνα Linux. Στην συνέχεια δημιουργούμε το σύστημα αρχείων πάνω από την καινούργια συσκευή, το οποίο μέσω των τροποποιήσεων θα γνωρίζει το πως αυτή διαμορφώθηκε. Ο οδηγός που διαλέγουμε να αξιοποιήσουμε είναι ο device mapper, που αποτελεί framework για απεικόνιση πολλαπλών block devices σε αφηρημένα, λογικά block devices [12]. Μπορούμε να κάνουμε ποικιλότροπα την απεικόνιση, με τις πιο γνωστές μεθόδους να αποτελούν η γραμμική και η τμηματική ένωση συσκευών (dm-linear και dm-stiped mapping αντιστοίχως [10]).

Μεταξύ των διαθέσιμων μεθόδων συνένωσης των συσκευών, επιλέγουμε την γραμμική, αφού αυτή προσφέρει σαφή διαχωρισμό μεταξύ των συσκευών εισόδου στον device mapper. Αυτό διευκολύνει πολύ και την κωδικοποίηση του τρόπου συνένωσης στην λογική του συστήματος αρχείων, αφού αρκεί να καταγράψουμε τα διαχωριστικά σημεία των διάφορων συσκευών που συναποτελούν την γραμμική συσκευή. Προφανώς, θα μπορούσαμε μέσω μιας τμηματικής συνένωσης να αξιοποιούμε σε κάποιο βαθμό το αθροιστικό bandwidth των συσκευών όλων των κόμβων, καθώς για κάθε πρόσβαση μεγέθους μεγαλύτερου του τμή-

ματος θα είχαμε εξυπηρέτηση της από περισσότερες συσκευές ταυτοχρόνως. Αυτό όμως επιβάλλει την ομοιόμορφη αξιοποίηση των συσκευών, άρα και μόνιμη επιβάρυνση του δικτύου διασύνδεσης των επεξεργαστών, και δεν παρέχει κάποιο εύχρηστο μέσο για την αντιμετώπιση των παθογενειών που αναφέρθηκαν στο κεφάλαιο *Πειράματα*. Στο κεφάλαιο *Αξιολόγηση* θα πραγματοποιηθούν συγκρίσεις της υλοποίησης μας με την επίδοση του τμηματικού τρόπου συνένωσης.

Βάσει των όσων αναφέραμε έως τώρα, λαμβάνουμε την συνολική εικόνα που παρουσιάζεται στο σχήμα 6.1. Μένει να προσδιορίσουμε κάποιες από τις λεπτομέρειες της διαδικασίας συνένωσης των συσκευών `/dev/pmem<X>`: Όπως θα δούμε στις τροποποιήσεις επί του συστήματος αρχείων, μονάδα αναζήτησης για την δέσμευση χώρου, τουλάχιστον σε αρχικό στάδιο, αποτελεί το block group. Επομένως, για να μπορούμε να επιλέγουμε κόμβο δέσμευσης αποδοτικά, αρκεί να αναθέσουμε κάθε block group σε έναν από τους επιμέρους κόμβους. Προκειμένου όμως η αντιστοίχιση να είναι με φυσικό τρόπο μονοσήμαντη, φροντίζουμε να μην είναι διαμοιρασμένο κάποιο block group σε διευθύνσεις διαφορετικών κόμβων. Ο άμεσος τρόπος να το επιτύχουμε αυτό είναι να φροντίσουμε το μέγεθος που αξιοποιούμε από κάθε συσκευή να είναι πολλαπλάσιο του μεγέθους ενός block group.

```
block_size_bytes=4096
blocks_per_group=32768
bytes_in_group=$(( block_size_bytes * blocks_per_group ))
sector_size_bytes=512

pmem0=/dev/pmem0
size_0=`blockdev --getsz $pmem0`
pmem0_bytes=$(( sector_size_bytes * size_0 ))
# Trim the size of each device to be perfectly divisible
# in terms of groups. The function round_down is defined
# in the script
size_0=$(( $(round_down pmem0_bytes bytes_in_group) /
    ↪ sector_size_bytes ))

# Do the same for pmem1 ...

# Make the linear mapping
echo "0 $size_0 linear $pmem0 0
$size_0 $size_1 linear $pmem1 0" | dmsetup create linear
```

Παράδειγμα διαμόρφωσης της ενιαίας συσκευής, με στρογγυλοποίηση του μεγέθους των επιμέρους συσκευών στο μέγεθος του block group

Η διαδικασία δημιουργίας της ενιαίας συσκευής φαίνεται στο αρχείο `experiments/scripts/mount/mount_create_linear.sh` του πηγαίου κώδικα της εργασίας. Για να αποφύγουμε την χρήση hard-coded τιμών,

δημιουργούμε μια φορά την συσκευή ώστε να λάβουμε μέσω του `dumpe2fs` τις τιμές `block_size_bytes` και `blocks_per_group`. Ύστερα, την δημιουργούμε εκ νέου έχοντας στρογγυλοποιήσει κατάλληλα τα μεγέθη των συσκευών `/dev/pmem<X>`. Το αρχείο που προκύπτει αναπαριστάται από το αρχείο συσκευής `/dev/mapper/linear`.

Είναι εύλογο το ερώτημα αν είναι καλή η πρακτική μη αξιοποίησης του χώρου που ψαλιδίζεται σε κάθε συσκευή, εφόσον οδηγεί σε μη ομοιόμορφη χρησιμοποίηση και μείωση της ανθεκτικότητας του χώρου των συσκευών. Ωστόσο, όπως συνηθίζεται σε σύγχρονα μέσα αποθήκευσης, οι μη πτητικές μνήμες διαθέτουν μηχανισμό για wear-leveling μέσω internal address translation [30], ο οποίος θεωρούμε ότι πιθανώς να αναιρεί το πρόβλημα που περιγράψαμε προηγουμένως.

6.2 Τροποποιήσεις στο Ext4

Όπως είδαμε στην ενότητα *Πειράματα*, η παρουσία απομακρυσμένων προσβάσεων προς τις μη πτητικές μνήμες μπορεί να συντελέσει αισθητά στην υποβάθμιση της επίδοσης τους. Το πρώτο και σημαντικότερο βήμα μας προς την παράκαμψη αυτού του προβλήματος είναι η εισαγωγή της έννοιας της NUMA τοπικότητας στις δεσμεύσεις δεδομένων όπως αυτές επιτελούνται από το σύστημα αρχείων που μας ενδιαφέρει, εν προκειμένω το ext4. Σκοπός της ενότητας αυτής είναι να παρουσιαστούν οι σχετικές τροποποιήσεις που πραγματοποιήθηκαν προς την κατεύθυνση αυτή. Αναφέρονται στις αμέσως επόμενες παραγράφους κάποια γενικά στοιχεία σχεδιασμού του ext4 στα οποίο στηριζόμαστε για την διαμόρφωση των εν λόγω τροποποιήσεων.

Κατά την δημιουργία και την επέκταση ενός καινούργιου αρχείου έχουμε δύο στάδια: ονομαστικά, αυτά είναι η δέσμευση ενός inode για το αρχείο, και η δέσμευση (πολλαπλών) block για τα δεδομένα του όταν αυτό είναι αναγκαίο. Και στις δύο περιπτώσεις, η αρχική μονάδα αναζήτησης για την δέσμευση χώρου στο μέσο είναι το block group, αντί για μεμονωμένα block. Η αναζήτηση σε επίπεδο block group επιτρέπει αφενός μεν την ευκολότερη αξιολόγηση της καταλληλότητας μιας περιοχής του μέσου αποθήκευσης για δέσμευση πολλαπλών block, αφετέρου δε την γρηγορότερη αναζήτηση και ενημέρωση δομών που σχετίζονται με την διαθεσιμότητα των block. Επομένως, για τους σκοπούς των τροποποιήσεων που θα περιγράψουμε χρειάζεται αρχικά να διαχωρίσουμε τα block group που ορίζονται από το ext4 στους επιμέρους NUMA κόμβους. Αργότερα, θα ανατρέχουμε στον διαχωρισμό αυτό για να επιλέγουμε block group για τις δεσμεύσεις που αντιστοιχούν στον ζητούμενο κάθε φορά NUMA κόμβο.

Το ext4 από σχεδιασμού του σέβεται την ανάγκη για τοπικότητα των μεταδεδομένων και δεδομένων ενός αρχείου. Αυτό είναι απτό αν αναλογιστούμε

τα χαρακτηριστικά των απλών μαγνητικών δίσκων, στους οποίους η διαδοχική ανάγνωση δεδομένων που δεν χαρακτηρίζονται από εγγύτητα επί του μέσου είναι ικανή να οδηγήσει σε μεγαλύτερες περιστροφές των μαγνητικών δίσκων, άρα και καθυστέρηση στην μεταφορά της ζητούμενης πληροφορίας. Αυτός ο σχεδιασμός έχει νόημα ακόμα και για πιο σύγχρονα μέσα αποθήκευσης, εφόσον πολλοί δίσκοι σταθερής κατάστασης (SSD), αλλά και οι μη πτητικές μνήμες (PMEM) που εξετάζουμε, διαθέτουν προσωρινές μνήμες για ομαδοποίηση των προσβάσεων [30]. Συνεπώς, η χωρική τοπικότητα (μετα)δεδομένων μπορεί να οδηγήσει σε καλύτερη αξιοποίηση αυτών των δομών.

Βάσει των πλεονεκτημάτων της χωρικής τοπικότητας που περιγράψαμε μόλις, οι τροποποιήσεις που θα πραγματοποιήσουμε στο ext4 θα έχουν ως γνώμονα την διατήρηση της ιδιότητας της τοπικότητας τόσο μεταξύ δεδομένων και μεταδεδομένων του ίδιου αρχείου, αλλά και μεταξύ διαφορετικών αρχείων όταν αυτό είναι αναγκαίο, κατά κύριο λόγο σε περίπτωση που αυτά βρίσκονται κάτω από τον ίδιο κατάλογο. Η προαναφερθείσα αρχή μπορεί να παραβιαστεί μόνο όταν έχουμε επέκταση ενός αρχείου από διεργασία διαφορετικού NUMA κόμβου, αφού αναγνωρίζουμε ότι η κατά NUMA τοπικότητα των εγγραφών είναι ύψιστης σημασίας για την επίδοση.

Σημειώνεται ότι στην συνέχεια της παρούσας ενότητας θα γίνει μια απόπειρα εποπτικής παρουσίασης των σχεδιαστικών επιλογών που έγιναν κατά την επέκταση του επιλεγμένου συστήματος αρχείων. Αυτό σημαίνει ότι επιθυμητή ιδιότητα του κειμένου αποτελεί περισσότερο να αιτιολογηθούν τα πιο κομβικά σημεία της υλοποίησης, και όχι απαραίτητα αυτή να περιγραφεί με πληρότητα. Η διευκρίνηση αυτή στοχεύει να αιτιολογήσει την ανομοιομορφία στον βαθμό εμβάθυνσης σε διάφορα σημεία της υλοποίησης. Για παράδειγμα, ενώ θα παρατεθεί μια αρκετά αναλυτική περιγραφή του κώδικα που προστέθηκε ή τροποποιήθηκε στα αρχεία κεφαλίδας `numa.h` και `ext4.h`, η ανάλυση των αλλαγών που πραγματοποιήθηκαν στις ρουτίνες δέσμευσης (μετα)δεδομένων θα είναι πιο αφαιρετική.

6.2.1 Προσθήκες στο αρχείο `ext4.h`

Θα εξετάσουμε αρχικά τις τροποποιήσεις που έγιναν στο βασικό αρχείο κεφαλίδας (header file) του πηγαίου κώδικα του συστήματος αρχείων ext4, ονόματι `ext4.h`. Στόχοι τους είναι:

Σ1. Να προσδιοριστούν καινούργιες σταθερές και δομές που θα εμπεριέχουν την απαραίτητη πληροφορία για την NUMA τοπολογία (αριθμός κόμβων και κατανομή των block group σε αυτούς)

Σ2. Να καταστήσουν εφικτή την προσαρμογή των βελτιστοποιήσεων της ε-

νότητας *Προϋπάρχουσες βελτιστοποιήσεις του Ext4* του κεφαλαίου *Θεωρητικό Υπόβαθρο* στις ανάγκες του NUMA awareness.

Ο στόχος [Σ1] επιτυγχάνεται εν μέρει κωδικοποιώντας τον αριθμό των NUMA κόμβων του συστήματος μέσω ενός συμβόλου του προεπεξεργαστή (`#define EXT4_NUMA_NUM_NODES 2`). Αυτή η πρακτική παρουσιάζει προβλήματα φορητότητας του μεταγλωττισμένου πυρήνα μεταξύ συστημάτων με διαφορετικό αριθμό κόμβων. Παραβλέπουμε ωστόσο αυτό το μειονέκτημα, δεδομένου ότι ο σκοπός της εργασίας είναι κατά κύριο λόγο η κατασκευή ενός proof of concept παρά μιας ολοκληρωμένης υλοποίησης. Επιπλέον, η ορθή μέθοδος αναγνώρισης του αριθμού των κόμβων εντός της λογικής του ext4 παρουσιάζει σημαντικές περιπλοκές στην συγγραφή του κώδικα.

Για να ολοκληρώσουμε τα απαιτούμενα του [Σ1], δημιουργούμε μια καινούργια δομή (struct) με όνομα `ext4_numa_info`. Αυτή περιέχει τις εξής πληροφορίες:

I) τον συνολικό αριθμό NUMA κόμβων του συστήματος: Παρόλο που ταυτίζεται πρακτικά με το `EXT4_NUMA_NUM_NODES`, διευκολύνει ως προς την αναγνωσιμότητα του κώδικα να υπάρχει ως μεταβλητή της δομής

II) το πρώτο block group κάθε κόμβου

III) τον συνολικό αριθμό από block group για κάθε κόμβο

Ως προς τις πληροφορίες των (II) και (III), έχουν γίνει δύο σημαντικές υποθέσεις: Η πρώτη είναι ότι η συνένωση των μη πτητικών συσκευών στην κατασκευή της γραμμικής συσκευής γίνεται με σεβασμό στην διάταξη των NUMA κόμβων. Για παράδειγμα, τις χαμηλότερες διευθύνσεις καταλαμβάνουν οι συσκευές του NUMA κόμβου 0, τις αμέσως χαμηλότερες διευθύνσεις καταλαμβάνουν οι συσκευές του κόμβου 1 κ.ο.κ. Αυτό είναι εύκολο να το διαβεβαιώσουμε μέσω του device mapper, και το υιοθετούμε ως υπόθεση γιατί διευκολύνει αρκετά την υλοποίηση.

Η δεύτερη υπόθεση είναι ότι δεν είναι απίθανο κάποιο block group που ορίζεται επί της γραμμικής συσκευής να είναι μοιρασμένο μεταξύ συσκευών διαφορετικών NUMA κόμβων. Όπως θα δούμε στην συνέχεια, διαχωρίζουμε τα block group της γραμμικής συσκευής μεταξύ των NUMA κόμβων στηριζόμενοι στην ισχύ της πρώτης υπόθεσης και στο γεγονός ότι οι συσκευές `/dev/pmem<X>` που συνενώνουμε έχουν το ίδιο ακριβώς μέγεθος. Συνεπώς, αν διαθέτουμε για παράδειγμα περιττό αριθμό από συνολικά block group σε ένα σύστημα με άρτιο αριθμό κόμβων, βλέπουμε ότι είναι βέβαιη η ύπαρξη διαμοιραζόμενου block group μεταξύ NUMA κόμβων.

Αναγνωρίζοντας την ισχύ της δεύτερης υπόθεσης, οδηγούμαστε στην ανάγκη διαμόρφωσης ενός κανόνα επίλυσης ζητημάτων διαμοιρασμού block group,

ο οποίος θα το αναθέτει σε έναν και μόνο κόμβο. Ο απλός κανόνας που επιλέχτηκε είναι να αναθέτουμε το διαμοιραζόμενο block group στον κόμβο με το μικρότερο αναγνωριστικό αριθμό. Για παράδειγμα, αν έχουμε ένα block group που μοιράζεται μεταξύ των NUMA κόμβων 0 και 1, αυτό θα ανατεθεί στον κόμβο 0. Η εναλλακτική λύση που επιτρέπει να θεωρούμε το block group κοινόχρηστο μεταξύ των δύο κόμβων εισάγει σημαντικές περιπλοκές στην υλοποίηση, και για αυτό την απορρίπτουμε. Σημειώνεται ωστόσο ότι γίνεται προσπάθεια εξάλειψης των διαμοιραζομένων block group κατά την κατασκευή της γραμμικής συσκευής, όπως είδαμε στην ενότητα *Ενοποίηση των μη πτητικών συσκευών*.

Βάσει των παραπάνω, καταλαβαίνουμε ότι δεν μπορούμε να εξασφαλίσουμε με απόλυτη βεβαιότητα πως όλοι οι NUMA κόμβοι θα έχουν τον ίδιο αριθμό από block group. Για αυτό, αποθηκεύουμε τον αριθμό των block group ανά κόμβο στο πεδίο (III), αντί να υποθέσουμε απλά ότι όλοι οι κόμβοι έχουν τον ίδιο αριθμό από block group, περίπτωση στην οποία θα μπορούσαμε να αξιοποιήσουμε απλά μια μόνο μεταβλητή.

Οι πληροφορίες της δομής `ext4_numa_info` αρχικοποιούνται μέσω της καινούργιας συνάρτησης `ext4_numa_super_init`. Είναι αντιληπτό ότι θα συμβουλευόμαστε την καινούργια δομή αρκετά συχνά, εφόσον αποσκοπούμε στο να κάνουμε δεσμεύσεις που σέβονται την κατά NUMA τοπικότητα. Είναι αναγκαίο λοιπόν να τοποθετήσουμε την αρχικοποιημένη δομή σε μια άλλη, προϋπάρχουσα δομή του ext4 που θα κάνει τις πληροφορίες αυτές διαθέσιμες κατά την εκτέλεση το αμεσότερο δυνατό. Η κατάλληλη δομή για να κάνουμε την ζητούμενη τοποθέτηση φαίνεται να είναι η `ext4_sb_info`, η οποία εμπεριέχει όλες τις πληροφορίες που είναι αποθηκευμένες στην μνήμη ("data in memory") κατά την λειτουργία του ext4.

Επανερχόμαστε στον πρόλογο της υποενότητας αυτής, στον οποίο αναφέραμε δύο στόχους για τις τροποποιήσεις του αρχείου `ext4.h`. Μεταφερόμαστε στην ανάλυση του στόχου [Σ2]. Υπενθυμίζεται ότι πρόκειται για την προσαρμογή των προϋπαρχουσών βελτιστοποιήσεων του ext4 στις ανάγκες του NUMA awareness. Οι βελτιστοποιήσεις αυτές, και οι ανάλογες προσαρμογές που έγιναν, είναι οι εξής:

A) **Ανά inode λίστα με preallocations**: Αντί να έχουμε μία μόνο λίστα `i_prealloc_list` ανά inode στην δομή `ext4_inode_info`, έχουμε συστοιχία `EXT4_NUMA_NUM_NODES` τέτοιων λιστών, από μια δηλαδή για κάθε κόμβο.

B) **Stream allocation:** Αντίστοιχα με προηγουμένως, ορίζουμε πλέον ανά κόμβο τα πεδία `s_mb_last_group` και `s_mb_last_start` της δομής `ext4_sb_info`. Τα πεδία αυτά συμβάλουν στην χωρική συνέχεια των δεσμεύσεων που γίνονται σε λειτουργία stream allocation, και αυτή αναμενόμενα θέλουμε να επιβάλλεται ξεχωριστά σε κάθε κόμβο.

Σε αυτό το σημείο αναδεικνύεται και η προγραμματιστική ευκολία που προέρχεται από την επιλογή ανάθεσης της πληροφορίας του αριθμού των NUMA κόμβων σε σύμβολο του προεπεξεργαστή. Διαφορετικά, για τα παραπάνω πεδία θα χρειαζόμασταν δυναμική δέσμευση μνήμης, η οποία θα πρόσθετε ανεπιθύμητο overhead και δυνητικά θα μπορούσε να οδηγήσει σε πιο σποραδικές προσβάσεις μνήμης.

Είναι απαραίτητο να σημειωθεί για το (A) ότι κατά την συγγραφή του κώδικα έγινε η υπόθεση πως η ανάθεση ενός κλειδώματος `i_prealloc_lock` ανά κόμβο δεν βλάπτει την ορθότητα του συστήματος αρχείων. Η υπόθεση αυτή βασίστηκε στο γεγονός ότι οι λίστες θεωρητικά δεν μπορούν να έχουν κοινά στοιχεία, εφόσον αναφέρονται σε δεσμεύσεις διαφορετικών κόμβων. Επιπροσθέτως, θεωρούμε ότι μπορεί να υπάρξει race condition μεταξύ πολλαπλών NUMA κόμβων που θέλουν να αξιοποιήσουν ο καθένας το δικό του preallocation για το ίδιο λογικό εύρος block ενός inode. Τέτοια συνθήκη επιλύεται νωρίς από κλειδώματα στις συναρτήσεις που πραγματοποιούν τις εγγραφές, οι οποίες αφορούν ολόκληρο το inode. Ωστόσο, λόγω περιορισμένης εξοικείωσης με θέματα συγχρονισμού, είναι πιθανό να χρειάζεται αναθεώρηση αυτός ο συλλογισμός.

Τέλος, αναφέρουμε μια ακόμα αλλαγή που δεν αφορά κάποια βελτιστοποίηση από αυτές που είδαμε στην θεωρία, αλλά μια συγκεκριμένη βοηθητική παράμετρο στην διαδικασία του inode allocation. Ο λόγος γίνεται για το πεδίο `i_last_alloc_group` της δομής `ext4_inode_info`, το οποίο επίσης σταματάει να είναι ενιαίο. Το πεδίο αυτό βοηθάει σε περίπτωση που το inode αναλογεί σε directory, ώστε να γνωρίζουμε σε ποιο block group έγινε τελευταία φορά δέσμευση μεταδεδομένων για κάποιο παιδί του. Στην επόμενη δέσμευση δοκιμάζεται πρώτα το καταγεγραμμένο block group, ευνοώντας έτσι την χωρική τοπικότητα. Με την αλλαγή που πραγματοποιήθηκε, εξακολουθεί να ισχύει αυτή η ιδιότητα, αλλά ως προς κάθε `NUMA` κόμβο.

6.2.2 Τα καινούργια αρχεία `numa.h` και `numa.c`

Στα νεοεισαχθέντα αρχεία `numa.h` και `numa.h` επιχειρούμε να κρύψουμε μεγάλο μέρος της επαναλαμβανόμενης λογικής που αφορά τον περιορισμό της αναζήτησης των block group σε κάποιον επιλεγμένο κόμβο. Είναι αναμενόμενο ότι σημαντικό μέρος της δουλειάς επί του ήδη υπάρχοντος κώδικα του ext4 είναι κατά προσέγγιση ο επόμενος γενικός μετασχηματισμός. Αυτός επιδέχεται πολλές παραλλαγές, οι οποίες εμφανίζονται συχνά στον ήδη υπάρχων κώδικα του ext4.

```
group = <some initial group>
ngroups = <total number of groups>
```

```
for (i = 0; i < ngroups; i++) {
    // work to be executed
    group = (group + step) % ngroups;
}
```

Πριν τον μετασχηματισμό: Αναζήτηση σε όλα τα group.

```
initial_node = <NUMA node where this is executed>

for_each_numa_node(iterator_node, initial_node,
    ↪ total_number_of_nodes) {
    // How many groups I have for this node
    n_node_groups = ext4_numa_num_groups(iterator_node);
    // Map this group to the current node
    group = ext4_numa_map_any_group(group, iterator_node);
    // For each group of this node
    for (i = 0; i < n_node_groups; i++) {
        // work to be executed
        group = ext4_numa_map_group(group + step, iterator_node);
    }
}
```

Μετά τον μετασχηματισμό: Αναζήτηση των group ανά κόμβο, με προτεραιότητα στον τοπικό.

Προφανώς, έχουν γίνει κάποιες απλοποιήσεις παραπάνω, καθώς στον κώδικα υπάρχουν και άλλες παράμετροι που αφορούν την υλοποίηση των συναρτήσεων απεικόνισης. Με τον όρο "συνάρτηση απεικόνισης" εννοούμε μια συνάρτηση που λαμβάνει κάποιο block group εισόδου και το αντιστοιχεί μονοσήμαντα σε κάποιο block group ενός επιθυμητού NUMA κόμβου εισόδου της συνάρτησης.

Στεκόμαστε για αρχή στο γεγονός ότι έχουμε δύο συναρτήσεις απεικόνισης και όχι απλά μια. Η πρώτη, ονόματι `ext4_numa_map_any_group`, λειτουργεί για οποιοδήποτε δοθέν block group. Η άλλη συνάρτηση, με όνομα `ext4_numa_map_group`, αποτελεί μια πιο γρήγορη και απλοποιημένη εκδοχή της προηγούμενης, που βασίζεται στην υπόθεση ότι το δοθέν block group έχει αναγνωριστικό αποκλειστικά μεγαλύτερο ή ίσο σε σχέση με το πρώτο group του επιθυμητού κόμβου. Η διάκριση της συγκεκριμένης περίπτωσης είναι ιδιαίτερα χρήσιμη, αφού πολύ συχνά μόνο αυξάνουμε το τρέχον group κατά την αναζήτηση, και μας ενδιαφέρει απλά να προλάβουμε την περίπτωση υπερχείλισης σε άλλον κόμβο.

Συναρτήσεις όπως αυτές της απεικόνισης, που ανήκουν σε μια ευρύτερη κατηγορία καινούργιων συναρτήσεων με πρόθεμα `ext4_numa`, μαζί με το σύμβολο `for_each_numa_node`, εμπεριέχονται στο νέο αρχείο `numa.h`. Οι συναρτήσεις περιορισμένου μεγέθους, όλες δηλαδή εκτός της `ext4_numa_super_init`, ορίζονται στο προαναφερθέν αρχείο κεφαλίδας ως `static inline`. Έχουμε επί-

γνωση του γεγονότος ότι αυτό δεν συνιστά απαραίτητα καλή προγραμματιστική πρακτική, ειδικά από άποψη αποσφαλμάτωσης του κώδικα. Ο σκοπός της όμως έγκειται στην μείωση του overhead από τις συχνές κλήσεις συναρτήσεων που πιθανώς δεν θα μπορούσε να κάνει `inline` ο μεταγλωττιστής της C από μόνος του.

Επιπροσθέτως, θέλουμε οι αλλαγές που αφορούν το NUMA awareness να ενεργοποιούνται κατά βούληση του διαχειριστή όταν προβαίνει σε προσάρτηση (mounting) του δίσκου που έχει διαμορφωθεί με την τροποποιημένη εκδοχή του συστήματος αρχείων ext4. Αυτό φροντίσαμε να ρυθμίζεται μέσω της παραμέτρου `-o numa`. Αυτή η ευελιξία θα μας εξυπηρετήσει σε μεγάλο βαθμό κατά την σύγκριση μεταξύ της αρχικής και της τροποποιημένης εκδοχής του, καθώς δεν θα χρειαστεί να ορίσουμε διαφορετικό σύστημα αρχείων που εμπεριέχει τις αλλαγές, ή να κάνουμε επανεκκίνηση με άλλον κάθε φορά πυρήνα για διαφορετικά υποσύνολα των επιθυμητών μετρήσεων. Έχοντας οργανώσει λοιπόν την επαναχρησιμοποιούμενη λογική για το NUMA awareness στις συναρτήσεις του αρχείου `numa.h`, καταφέρνουμε να έχουμε πιο οργανωμένο έλεγχο ως προς το αν θέλουμε να χρησιμοποιηθεί ο καινούργιος κώδικας ή όχι: αρκεί στην αρχή των αντίστοιχων συναρτήσεων να καλέσουμε την συνάρτηση `ext4_numa_enabled`, και να προσαρμόζουμε το αποτέλεσμα της καλούσας συνάρτησης αναλόγως.

Αναφέρουμε ως παράδειγμα του προαναφερθέντος σχεδιασμού ότι αντί να χρησιμοποιήσουμε απευθείας στον κώδικα που αφορά τις δεσμεύσεις (μετα)δεδομένων το σύμβολο `EXT4_NUMA_NUM_NODES`, αξιοποιούμε απεναντίας την συνάρτηση `ext4_numa_num_nodes`. Μέσω αυτής είτε επιστρέφεται η τιμή του προαναφερθέντος συμβόλου σε περίπτωση που έχει δοθεί η επιλογή `-o numa`, είτε επιστρέφεται μονάδα σε διαφορετική περίπτωση. Θεωρούμε δηλαδή κατά σύμβαση ότι έχουμε μόνο έναν κόμβο. Με αυτόν τον τρόπο, αν συναντήσουμε στην συνέχεια το σύμβολο `for_each_numa_node(..., num_nodes)` και κατά την προσάρτηση δεν είχε δοθεί η παράμετρος `-o numa`, θα πραγματοποιηθεί μια μόνο επανάληψη. Έτσι, στην πραγματικότητα αναιρείται η επίδραση που θα είχε ο επιπλέον κώδικας για το NUMA awareness.

Πρέπει να σημειωθεί ότι ιδανικά, τόσο από άποψη πληρότητας της εργασίας όσο και ως απόδειξη ορθότητας του σχεδιασμού των αλλαγών που πραγματοποιήθηκαν, θα χρειαζόταν να πραγματοποιηθεί τυπική απόδειξη για την ικανότητα του τροποποιημένου συστήματος αρχείων να συμπεριφέρεται πανομοιότητα με την μη τροποποιημένη εκδοχή του ελλείψει της παραμέτρου `-o numa`. Ωστόσο, αυτό είναι αρκετά περίπλοκο ως διαδικασία και πιθανώς ξεφεύγει του σκοπού της διπλωματικής εργασίας, οπότε αρκούμαστε στην σχετική επιμέλεια που υπήρξε στην διάρκεια συγγραφής του κώδικα και τα θετικά αποτελέσματα που παρατηρήθηκαν κατά τις άτυπες διαδικασίες ελέγχου.

Στην συνέχεια, παρέχουμε μια επισκόπηση και σχολιασμό των σημαντικό-

τερων συναρτήσεων και συμβόλων που διατίθενται μέσω του αρχείου `numa.h`. Υποθέτουμε ότι έχει δοθεί η παράμετρος `-o numa` κατά την προσάρτηση, αλλιώς οι τιμές που επιστρέφονται συμβαδίζουν με την θεωρητική υπόθεση ότι έχουμε μόνο έναν NUMA κόμβο.

- `for_each_numa_node(_n, var_node, init_node, num_nodes)`

  Είμαστε αναγκασμένοι να συμπεριλάβουμε στον ορισμό του συμβόλου την βοηθητική μεταβλητή `_n`, αν και είναι λεπτομέρεια της υλοποίησης της αντίστοιχης επανάληψης `for`. Διαφορετικά, αν δηλώσουμε την μεταβλητή εντός της `for`, λαμβάνουμε προειδοποίηση κατά την μεταγλώττιση. Επίσης, παρέχεται ως παράμετρος ο αριθμός των NUMA κόμβων ακριβώς για τους λόγους που παρουσιάσαμε λίγες παραγράφους πριν.

- `bool ext4_numa_enabled(struct super_block *sb)`

  Ελέγχει αν έχει δοθεί η παράμετρος `-o numa` κατά την προσάρτηση του δίσκου που περιέχει το σύστημα αρχείων. Χρησιμοποιείται σχεδόν σε όλες τις επόμενες συναρτήσεις.

- `int ext4_numa_node_id(struct super_block *sb)`

  Επιστρέφει το αναγνωριστικό του NUMA κόμβου στον οποίο εκτελείται η συνάρτηση.

- `int ext4_numa_num_nodes(struct super_block *sb)`

  Επιστρέφει τον αριθμό των NUMA κόμβων του συστήματος (ταυτίζεται με την τιμή του συμβόλου `EXT4_NUMA_NUM_NODES`).

- `int ext4_numa_bg_node(struct super_block *sb,`
  `↪ ext4_group_t g)`

  Επιστρέφει τον NUMA κόμβο στον οποίο αντιστοιχεί το δοθέν block group.

- `ext4_group_t ext4_numa_map_group(struct super_block *sb,`
  `↪ ext4_group_t group, int map_node, ext4_group_t ngroups)`

  Επιστρέφει την απεικόνιση του group εισόδου `group` στον κόμβο εισόδου `map_node`. Υποθέτουμε ότι το δοθέν group ανήκει απαραίτητα είτε στον ζητούμενο κόμβο, είτε σε κάποιον με μεγαλύτερο αναγνωριστικό.

- `ext4_group_t ext4_numa_map_any_group(`
  `↪ struct super_block *sb, ext4_group_t group,`
  `↪ int map_node)`

  Όπως η προηγούμενη συνάρτηση, χωρίς όμως τον περιορισμό που αναφέρθηκε.

- `ext4_fsblk_t ext4_numa_map_any_block(`
  `↪ struct super_block *sb, ext4_fsblk_t block,`
  `↪ int map_node)`

  Όπως η `ext4_numa_map_any_group`, μόνο που η απεικόνιση γίνεται σε επίπεδο block αντί block group. Αυτό είναι χρήσιμο ως προς την εξασφάλιση της ευρωστίας ορισμένων αλλαγών που έγιναν στο σύστημα αρχείων. Συγκεκριμένα, στην περίπτωση που μας έχει δοθεί ένα block ως στόχος (όπως συμβαίνει σε ένα αίτημα δέσμευσης) το οποίο λόγω προηγούμενης αστοχίας δεν συμβαδίζει με την κατά NUMA τοπικότητα, μέσω της συνάρτησης αυτής σιγουρευόμαστε ότι θα απεικονιστεί στον επιθυμητό κόμβο.

- `int ext4_numa_block_node(struct super_block *sb,`
  `↪ ext4_fsblk_t blk)`

  Επιστρέφει τον NUMA κόμβο του δοσμένου block.

- `void ext4_numa_super_init(struct ext4_sb_info *sbi)`

  Καλείται μέσα στην `ext4_fill_super` (του αρχείου `super.c`) ώστε να αρχικοποιηθούν οι πληροφορίες της δομής `ext4_numa_info` που εμπεριέχεται στην δομή `ext4_sb_info`. Έχουν περιγραφεί ήδη στην προηγούμενη ενότητα κάποιες πολιτικές που αφορούν την διαδικασία συμπλήρωσης των σχετικών πεδίων, και οι λεπτομέρειες στην λογική της υλοποίησης ελπίζουμε πως καλύπτονται επαρκώς από τα σχετικά σχόλια που συνοδεύουν την συνάρτηση.

Με αυτήν την αναφορά, ο αναγνώστης ελπίζουμε πως είναι σε καλύτερη θέση να διαβάσει και να κατανοήσει ευκολότερα την λογική και τον σχεδιασμό των αλλαγών που έγιναν στο ext4, όπως αυτές θα περιγραφούν στην συνέχεια. Στις επόμενες δύο υποενότητες θα δούμε όσο πιο αφαιρετικά γίνεται τις τροποποιήσεις και προσθήκες που έγιναν για την εισαγωγή χαρακτηριστικών NUMA awareness, πρώτα στο inode allocation και ύστερα στο multiblock allocation.

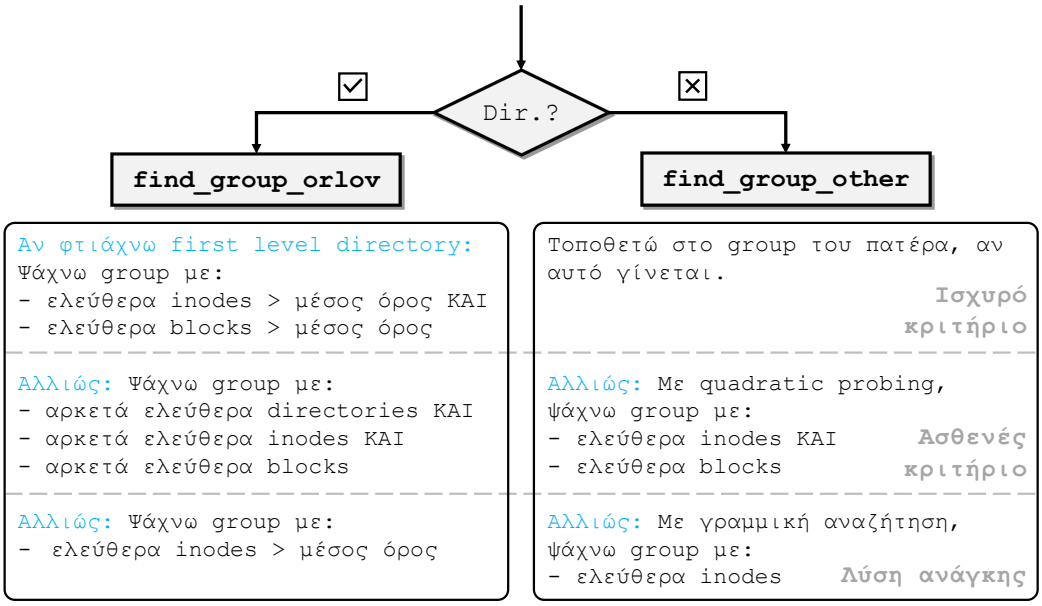
6.2.3 Inode Allocation

Κάνοντας ανάλυση μέσω των μεθόδων που παρουσιάστηκαν στην ενότητα *Tracing στον πυρήνα*, εντοπίζουμε την επόμενη αλληλουχία συναρτήσεων για την δέσμευση μεταδεδομένων στην περίπτωση δημιουργίας καινούργιου αρχείου.

```
do_sys_open          → do_sys_openat2    →
do_filp_open         → path_openat       →
open_last_lookups    → lookup_open       →
ext4_create (ptr)    → __ext4_new_inode
```

Παραπάνω, η σημείωση (ptr) διευκρινίζει ότι η αντίστοιχη συνάρτηση βρέθηκε μέσω κάποιου δείκτη. Αναφέρονται αυτές οι περιπτώσεις γιατί δεν είναι πάντα εύκολο να προσδιορίσουμε από που καλείται μια τέτοια συνάρτηση, τουλάχιστον από αναζήτηση με εργαλεία όπως το elixir bootlin. Από την παραπάνω αλληλουχία συναρτήσεων, εστιάζουμε στην συνάρτηση `__ext4_new_inode`, καθώς φαίνεται η πιο υποσχόμενη.

Στο σχήμα 6.2 έχουμε μια αφαιρετική, σχηματική παρουσίαση του inode allocation στην αρχική εκδοχή του ext4. Βλέπουμε ότι υπάρχει διαφοροποίηση στην συμπεριφορά της συνάρτησης δέσμευσης inode αναλόγως με το αν έχουμε αίτημα για inode ενός directory ή ενός αρχείου.



Σχήμα 6.2: Αφαιρετική αναπαράσταση της συνάρτησης `__ext4_new_inode` στην αρχική της εκδοχή

Είναι σημαντικό να αναφερθεί ότι κατά τις τροποποιήσεις που παρουσιάζονται στην συνέχεια δεν δόθηκε προτεραιότητα στην διατήρηση υποστήριξης του χαρακτηριστικού flex group (που είδαμε στην ενότητα *Προϋπάρχουσες βελτιστοποιήσεις του Ext4* του κεφαλαίου *Θεωρητικό Υπόβαθρο*). Για αυτόν τον λόγο η παράμετρος `-o numa` κατά την προσάρτηση ενός δίσκου με το τροποποιημένο σύστημα αρχείων οφείλει να συνοδεύεται από την παράμετρο `-o ^flex_bg`,

το οποίο και ελέγχεται στην συνάρτηση `ext4_fill_super`. Ενώ αυτό βλάπτει την πληρότητα των τροποποιήσεων, σκοπός ήταν να αξιοποιηθεί ο αντίστοιχος χρόνος για να γίνουν σωστά οι αλλαγές στο mutliblock allocation. Παρόλα αυτά, οι απαιτούμενες προσαρμογές για να διορθωθεί αυτή η έλλειψη δεν είναι υπερβολικά σύνθετες, και θα μπορούσαν να είναι μέρος μελλοντικής επέκτασης.

Στην περίπτωση ενός directory, εκτελείται ο λεγόμενος αλγόριθμος του Orlov. Στο σχήμα έχουν γίνει σημαντικές απλοποιήσεις, και δεν εμπλέκεται καθόλου η λογική της βελτιστοποίησης των flex group εφόσον αυτό δεν υποστηρίζεται από τις τροποποιήσεις που πραγματοποιήσαμε. Για περαιτέρω μελέτη της συμπεριφοράς του αλγορίθμου, παραπέμπουμε στα σχόλια που συνοδεύουν την συνάρτηση στον αρχικό κώδικα του πυρήνα 5.13, τα οποία είναι αρκετά κατατοπιστικά εφόσον περιγράφουν πλήρως τον σκελετό της ρουτίνας αυτής. Επίσης, όπως ισχύει και για τις υπόλοιπες περιπτώσεις συναρτήσεων που θα εξετάσουμε, οι αλλαγές που έγιναν για τους σκοπούς της εργασίας μπορούν να φανούν άμεσα μέσω της ακόλουθης εντολής:

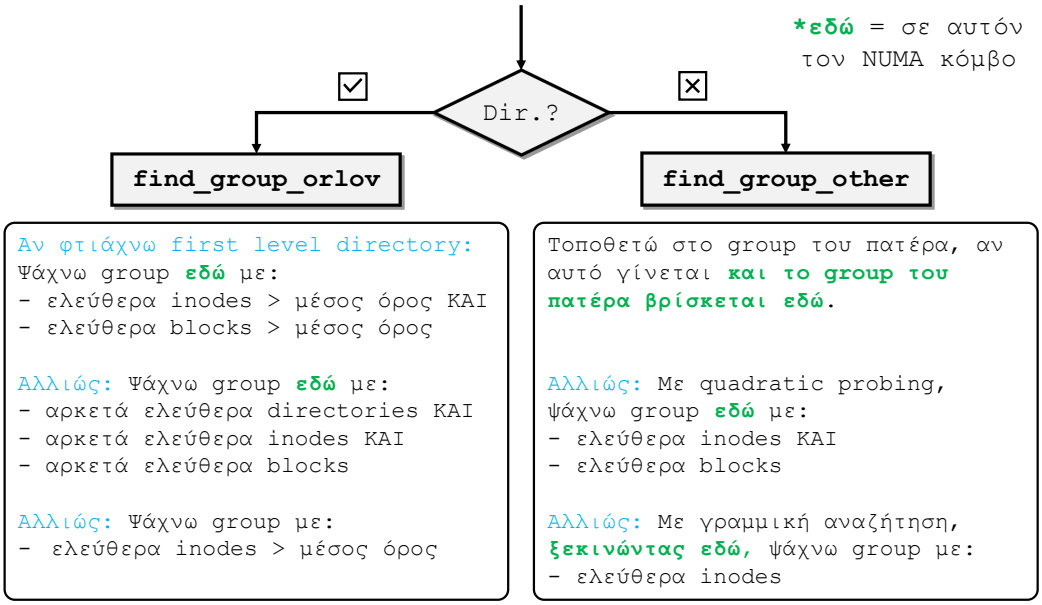
```
$ git diff 32debcbccb4c93a3f3b188e9e8eb15d5e9951e37 HEAD
   ↪ fs/ext4/ialloc.c
```

Η συμβολοσειρά `32debcbccb4c93a3f3b188e9e8eb15d5e9951e37` αντιστοιχεί στο hash του πρώτου commit, που βρίσκουμε μέσω της εντολής `git log`, και το `HEAD` αντιστοιχεί στο τελευταίο commit. Υπήρξε μέριμνα για να συνοδεύονται οι αλλαγές αυτές από τα απαραίτητα σχόλια για την καλύτερη κατανόηση τους.

Συνοπτικά, στόχος του αλγορίθμου είναι πρώτα να κατανέμει δίκαια μεταξύ των groups τις καταχωρήσεις στο πρώτο επίπεδο του directory tree, μέσω εξέτασης των ελεύθερων inode και block. Αυτό αποσκοπεί στην ομοιόμορφη αξιοποίηση του μέσου αποθήκευσης. Σε περίπτωση που η αναζήτηση απέτυχε ή δεν δόθηκε first level directory, κάνουμε γραμμική αναζήτηση των group ώστε να βρούμε ένα που ικανοποιεί μια σειρά αυστηρών κριτηρίων, μεταξύ των οποίων η διαθεσιμότητα αρκετών inode, καταλόγων αλλά και block. Αν η προηγούμενη αναζήτηση αποδειχθεί ανεπιτυχής, χαλαρώνουμε το κριτήριο της αναζήτησης, προβαίνοντας σε εύρεση του group με συγκριτικά περισσότερα ελεύθερα inode. Έχουμε λοιπόν δύο φάσεις του αλγορίθμου που ακολουθούν ένα αυστηρό σύνολο κριτηρίων, και μια τρίτη φάση που εκτελείται αν οι δύο προηγούμενες αποτύχουν.

Η πιο φυσική προσαρμογή του αλγόριθμου του Orlov για προσθήκη χαρακτηριστικών NUMA awareness είναι η εξής: Από την μια, περιορίζουμε την αναζήτηση των δύο πρώτων φάσεων στα group του τοπικού NUMA κόμβου (δηλαδή αυτού από τον οποίο καλείται η συνάρτηση). Από την άλλη, αφή-

νουμε την αναζήτηση της τρίτης φάσης ελεύθερη να επεκταθεί και σε άλλους κόμβους, μόνο όμως σε περίπτωση που διαπιστώσουμε ότι δεν υπάρχει κανένα κατάλληλο group διαθέσιμο στον τρέχοντα κόμβο. Θα δούμε ευθύς αμέσως ότι παρόμοια δομή έχουν και οι αλλαγές που έγιναν στην περίπτωση δέσμευσης μεταδεδομένων αρχείων.



Σχήμα 6.3: Αφαιρετική αναπαράσταση της συνάρτησης `__ext4_new_inode` μετά τις αλλαγές

Για την συνάρτηση `find_group_other` που αναλαμβάνει την εύρεση group για δέσμευση του inode ενός νέου αρχείου, έχουμε ελαφρώς διαφορετική λογική. Πρώτος παράγοντας που εξετάζουμε είναι αν μπορεί να τοποθετηθεί επιτυχώς το νέο inode στο group του πατέρα του, δηλαδή του καταλόγου στο οποίο υπάγεται το αντίστοιχο αρχείο. Σε περίπτωση που αυτό δεν είναι εφικτό οδηγούμαστε στο δεύτερο στάδιο του αλγορίθμου, στο οποίο αξιοποιούμε quadratic probing με είσοδο το group g του γονέα, προς εύρεση group που διαθέτει και ελεύθερα inodes και ελεύθερα blocks. Με τον όρο quadratic probing εννοούμε ότι στην επανάληψη i της αναζήτησης ελέγχουμε το group $(g + i^2)$ (σε αριθμητική υπολοίπου με διαιρέτη το πλήθος των group). Σκοπός του είναι ο πιθανός διαχωρισμός των αρχείων μη ταυτιζόμενων καταλόγων τα οποία τυχαίνει να μοιράζονται το ίδιο group. Αν η προηγούμενη αναζήτηση αποτύχει, προβαίνουμε στο τρίτο και τελευταίο στάδιο, στο οποίο ψάχνουμε γραμμικά τα group με το πιο χαλαρό κριτήριο της απλής διαθεσιμότητας ελεύθερων inode.

Όπως και στον αλγόριθμο του Orlov, και εδώ περιορίζουμε τις πρώτες δύο φάσεις στον τρέχοντα κόμβο, ενώ επιτρέπουμε στην τρίτη και τελευταία να

επεκτείνει την αναζήτηση και σε άλλους κόμβους αν κανένα group του τρέχοντος δεν κρίνεται κατάλληλο. Σημειώνεται ότι η συμπεριφορά του αλγορίθμου είναι διαφορετική αν έχουμε ενεργή την λειτουργία των flex group, καθώς τότε αν αποτύχει η αναζήτηση του γονέα καλούμε τον αλγόριθμο του Orlov με τις κατάλληλες παραμέτρους. Συνεπώς, για την προσθήκη συμβατότητας μεταξύ των νέων χαρακτηριστικών του NUMA awareness και των flex group, χρειάζεται ειδική μέριμνα ως προς αυτήν την περίπτωση.

6.2.4 Multiblock Allocation

Χρησιμοποιώντας τις μεθόδους της ενότητας *Tracing στον πυρήνα* λαμβάνουμε την ακόλουθη αλληλουχία κλήσεων για μια εγγραφή σε αρχείο που μπορεί να οδηγήσει στην δέσμευση block δεδομένων:

```
vfs_write                → new_sync_write        →
call_write_iter          → ext4_file_write_iter  →
ext4_dax_write_iter      → dax_iomap_rw          →
iomap_apply              → ext4_iomap_begin      →
ext4_iomap_alloc         → ext4_map_blocks       →
ext4_ext_map_blocks      → ext4_mb_new_blocks    →
ext4_mb_regular_allocator
```

Από ανάγνωση του κώδικα των σχετικών συναρτήσεων αναγνωρίζουμε ότι περισσότερο ενδιαφέρον για την προσθήκη χαρακτηριστικών NUMA awareness έχουν οι τρεις τελευταίες συναρτήσεις:

- `ext4_ext_map_blocks`
  Αναζήτηση στο extent tree για να διαπιστώσουμε αν μπορεί να αξιοποιηθεί κάποιο ήδη υπάρχον extent του αρχείου (για overwrite) ή κάποιο δεσμευμένο cluster που έχει επικάλυψη με τα κοντινά στο ζητούμενο λογικό εύρος extent του αρχείου (για append). Αν δεν πετύχουν οι σχετικοί έλεγχοι, καλούμε την `ext4_mb_new_blocks` ώστε να βρεθεί κατάλληλο αξιοποιήσιμο block.

- `ext4_mb_new_blocks`
  Πρώτα ελέγχουμε τον προδεσμευμένο χώρο ("preallocation space") που έχει προκύψει από βελτιστοποιήσεις που είδαμε στην ενότητα της θεωρίας. Αν δεν υπάρχουν κατάλληλα preallocation, κάνουμε εξ ολοκλήρου καινούργιες δεσμεύσεις μέσω της επόμενης συνάρτησης.

- `ext4_mb_regular_allocator`
  Ψάχνει ένα κατάλληλο block group, δηλαδή αυτό που θα προσφέρει την ιδανικότερη διαθέσιμη σειρά από block για τους σκοπούς της ζητούμενης δέσμευσης.

Σημειώνουμε ότι έχουμε δύο πιθανούς σχεδιασμούς για τον τρόπο επιλογής NUMA κόμβου κατά την επέκταση ενός αρχείου: ο πρώτος επιλέγει πάντα τον κόμβο στον οποίο βρίσκεται το νήμα που πραγματοποιεί την εγγραφή, ενώ ο δεύτερος επιλέγει τον κόμβο στον οποίο βρίσκεται το inode του αρχείου που επεκτείνουμε. Ο ένας σχεδιασμός ευνοεί την κατά NUMA τοπικότητα των προσβάσεων, ενώ ο δεύτερος ευνοεί την τοπικότητα των block δεδομένων του αρχείου. Βάσει των αποτελεσμάτων της ενότητας με τα πειράματα, διαλέγουμε να υλοποιήσουμε τον πρώτο σχεδιασμό. Ωστόσο, μια πιο έξυπνη υλοποίηση πιθανώς να ενσωμάτωνε κατά περιπτώσεις και τον δεύτερο.

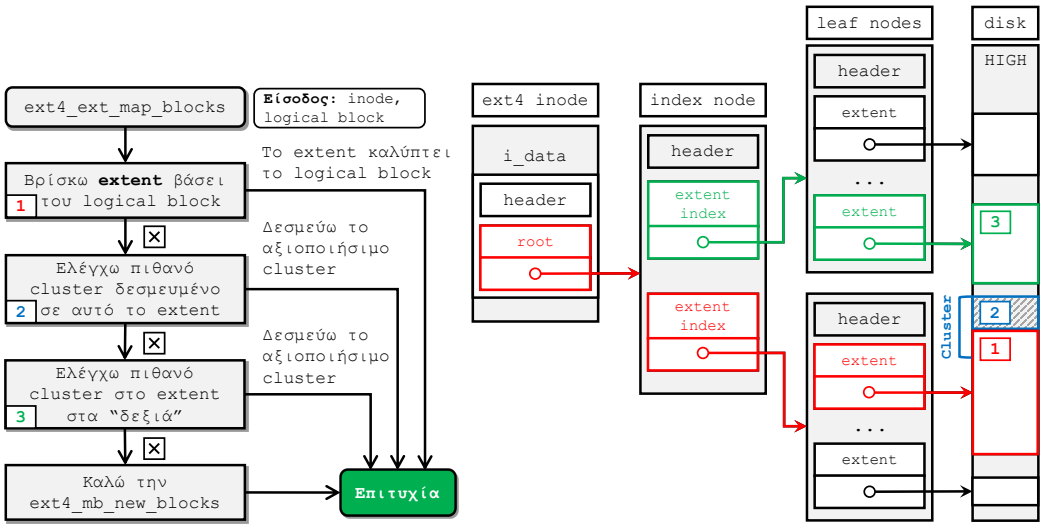
Προβαίνουμε σε μελέτη κάθε συνάρτησης ξεχωριστά, προκειμένου να προσδιορίσουμε τον τρόπο με τον οποίο μπορούμε να τις προσαρμόσουμε για τους σκοπούς της εργασίας.

Συνάρτηση ext4_ext_map_blocks

Διευκρινίζεται ότι με τους όρους "λογικό block" και αντίστοιχα "λογικό εύρος (block)" ενός αρχείου αναφερόμαστε στα block που είτε αυτό καταλαμβάνει ήδη είτε πρόκειται να δεσμεύσει, αριθμημένα με την σειρά που εμφανίζονται στο αρχείο. Για παράδειγμα, το λογικό block 0 ενός αρχείου αντιστοιχεί στα δεδομένα που βρίσκονται στην αρχή του, και μπορεί να αντιστοιχεί σε οποιοδήποτε block του συστήματος αρχείων, ανεξαρτήτως της λογικής αρίθμησης του. Το λογικό block 1 αντιστοιχεί στα αμέσως επόμενα δεδομένα του κλπ.

Ξεκινάμε με την συνάρτηση απεικόνισης του ζητούμενου εύρους λογικών block του αρχείου στο extent tree (`ext4_ext_map_blocks`). Όλα όσα θα περιγράψουμε αποδίδονται στο σχήμα 6.4. Πρώτα δοκιμάζουμε να δούμε αν υπάρχει ήδη extent που καλύπτει τα ζητούμενα λογικά block, περίπτωση η οποία αντιστοιχεί σε κλήση για overwrite. Εφόσον ενδιαφερόμαστε για την περίπτωση δέσμευσης καινούργιων block, προχωράμε στον επόμενο έλεγχο.

Σε αυτό το σημείο δεν έχουμε βρει κάποιο extent που καλύπτει ολόκληρο το ζητούμενο λογικό εύρος, αλλά έχουμε εντοπίσει τον πιο κοντινό φύλλο του extent tree, δηλαδή αυτό που αντιστοιχεί στο αμέσως μικρότερο δεσμευμένο λογικό εύρος. Αυτό το φύλλο (στην ουσία το αντίστοιχο extent) πιθανώς να δεσμεύει κάποιο cluster που να μην αξιοποιείται πλήρως. Όπως είπαμε στην ενότητα *Προϋπάρχουσες βελτιστοποιήσεις του Ext4* του κεφαλαίου της θεωρίας, μέσω του cluster allocation η μικρότερη δυνατή μονάδα δέσμευσης δεν είναι μεμονωμένα block, αλλά ομάδες από block ενός επιλεγμένου σταθερού αριθμού,



Σχήμα 6.4: Τρόπος λειτουργίας της συνάρτησης `ext4_ext_map_blocks`
Σημείωση: Στην περίπτωση (3) επί του σχήματος έχει χρωματιστεί το ίδιο το extent, αλλά εννοείται ότι απλά ελέγχουμε τα μερικώς αξιοποιημένα cluster αυτού, όπως στο βήμα (2).

τα λεγόμενα cluster. Με αυτόν τον τρόπο μπορεί κάποια στιγμή να δεσμεύσουμε περισσότερα block από όσα μας είναι απαραίτητα. Εξετάζουμε λοιπόν αν ισχύει αυτή ακριβώς η περίπτωση, ώστε να ικανοποιήσουμε μέρος του αιτήματος μέσω των διαθέσιμων block του cluster. Αν δεν ισχύει αυτό, κάνουμε τον ίδιο έλεγχο για το αμέσως επόμενο extent (αυτό στα "δεξιά") του τρέχοντος.

Αν έχουν αποτύχει όλοι οι προηγούμενοι έλεγχοι, φτάνουμε στην συνειδητοποίηση ότι πρέπει να αξιοποιήσουμε διαθέσιμα block που δεν αναλογούν στο extent tree, το οποίο επιχειρούμε μέσω της συνάρτησης `ext4_mb_new_blocks`.

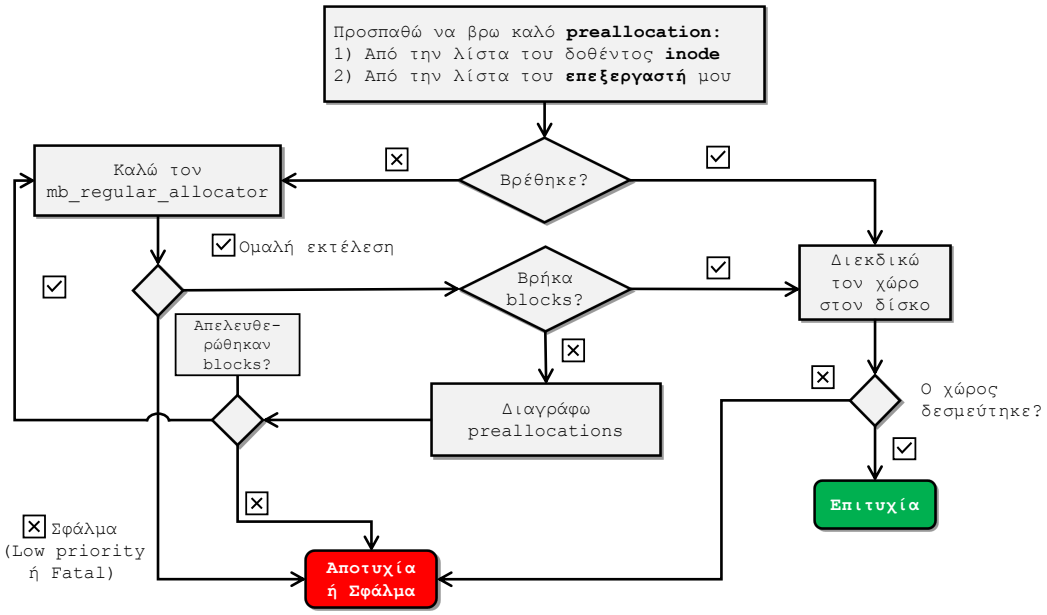
Επισημαίνουμε ότι τα βήματα (1), (2) και (3) του σχήματος 6.4, τα οποία αφορούν τον έλεγχο επί του extent tree την στιγμή της κλήσης της `ext4_ext_map_blocks`, ενέχουν τον κίνδυνο απομακρυσμένης δέσμευσης block. Ο λόγος είναι ότι μπορεί πρώτα να δεσμεύσουμε μερικώς κάποιο cluster ενός NUMA κόμβου, και ύστερα να βρεθεί ευκαιρία να το αξιοποιήσουμε περαιτέρω σε κλήση δέσμευσης των αμέσως επόμενων λογικών block του αρχείου από τελείως διαφορετικό κόμβο. Οφείλουμε να επιδιώξουμε την πλήρωση των cluster με αυτόν τον τρόπο, γιατί διαφορετικά η βελτιστοποίηση θα αρχίσει να αποφέρει μη αξιοποιήσιμα block στα δεσμευμένα clutser. Αναγνωρίζουμε λοιπόν ότι δεν μπορούμε να κάνουμε κάτι για αυτήν την περίπτωση προκειμένου να αποτρέψουμε την απομακρυσμένη δέσμευση, εκτός ίσως της απενεργοποίησης του cluster allocation. Αυτό όμως θεωρούμε ότι θα έχει σημαντική επίπτωση στην επίδοση του συστήματος αρχείων, και για αυτό το απορρίπτουμε ως ιδέα.

Καταλήγουμε με την συλλογιστική που προηγήθηκε στο συμπέρασμα ότι

δεν μπορούμε να μεταβάλουμε πολλά στην λειτουργία της συνάρτησης ext4_ext_map_blocks, για αυτό επικεντρωνόμαστε στον τρόπο με τον οποίο μπορούμε να προσαρμόσουμε την συνάρτηση ext4_mb_new_blocks ώστε να σέβεται την κατά NUMA τοπικότητα.

Συνάρτηση ext4_mb_new_blocks

Στο σχήμα 6.5 παρουσιάζεται μια απλοποιημένη σχηματική αναπαράσταση μέσω διαγράμματος ροής της συνάρτησης ext4_mb_new_blocks πριν τις τροποποιήσεις.



Σχήμα 6.5: Διάγραμμα ροής της συνάρτησης ext4_mb_new_blocks στην αρχική της εκδοχή

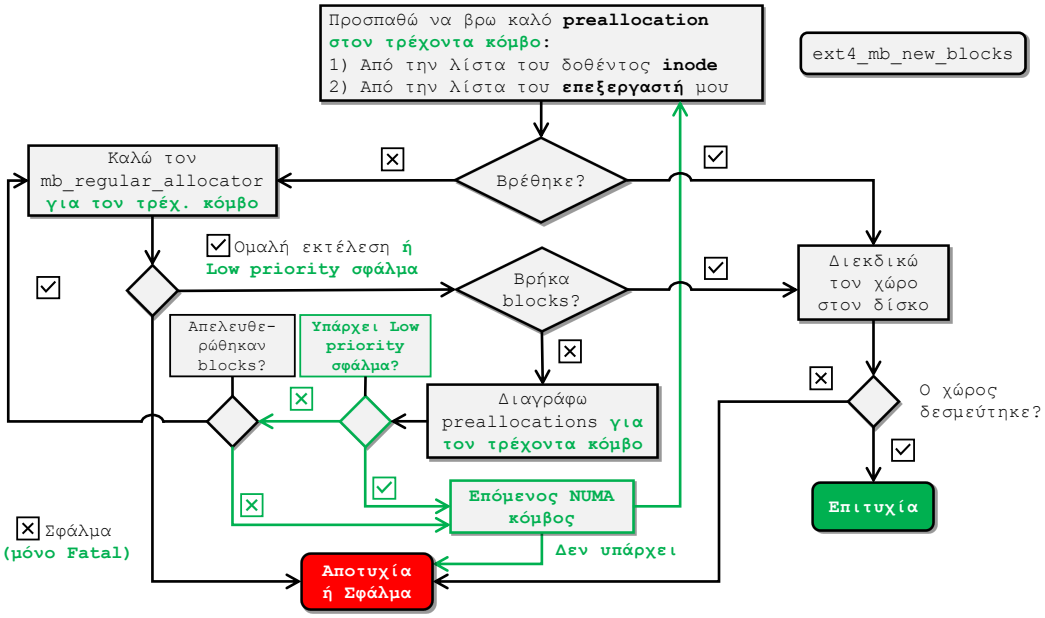
Πρώτο βήμα αποτελεί ο έλεγχος διαθεσιμότητας κάποιου αξιοποιήσιμου preallocation, μέσω κλήσης της συνάρτησης ext4_mb_use_preallocated. Αρχικά ελέγχουμε την λίστα i_prealloc_list του συγκεκριμένου inode. Αν αυτή διαθέτει preallocation που καλύπτει πλήρως το εύρος λογικών block του αιτήματος δέσμευσης (εκπεφρασμένο μέσω της παραμέτρου τύπου ext4_allocation_context της συνάρτησης ext4_mb_use_preallocated), τότε το αξιοποιούμε.

Αν δεν υπάρχει κατάλληλο preallocation του συγκεκριμένου inode, και το αίτημα είναι αρκετά μικρό (ως προς το μέγεθος δηλαδή του απαιτούμενου χώρου) ώστε να έχουμε hint για group allocation (flag EXT4_MB_HINT_GROUP_ALLOC), τότε εξετάζουμε το locality group του συγκεκριμένου επεξεργαστικού

πυρήνα στον οποίον εκτελείται η συνάρτηση. Αν υπάρχουν αξιοποιήσιμα preallocation, με την έννοια ότι καλύπτεται το μήκος του αιτήματος, επιλέγουμε αυτό που είναι πιο κοντά στο block-στόχο του δοθέντος αιτήματος.

Αν διαπιστώσουμε ότι δεν υπάρχει κάποιο κατάλληλο preallocation, προβαίνουμε στην δέσμευση καινούργιων cluster μέσω της κλήσης της συνάρτησης `ext4_mb_regular_allocator`. Σε περίπτωση που η προαναφερθείσα συνάρτηση δεν παρουσίασε κάποιο σφάλμα αλλά ταυτόχρονα δεν ικανοποίησε το δοθέν αίτημα, δοκιμάζουμε να διαγράψουμε όσα preallocation χρειάζονται ώστε να απελευθερωθεί ο απαιτούμενος από το αίτημα χώρος. Σε περίπτωση που αυτό πετύχει (η συνάρτηση `ext4_mb_discard_preallocations_should_retry` επιστρέφει αληθή τιμή), προσπαθούμε εκ νέου να καλέσουμε την συνάρτηση `ext4_mb_regular_allocator`.

Στο σχήμα 6.6 παρατίθεται το ίδιο διάγραμμα ροής, συμπεριλαμβανομένων αυτή την φορά των σχετικών τροποποιήσεων. Μια πρώτη προφανής αλλαγή είναι ο περιορισμός της αναζήτησης των preallocation στον NUMA κόμβο όπου πραγματοποιείται η εκτέλεση της συνάρτησης. Θεωρητικά αυτό ακούγεται αρκετά απλό ως προς την υλοποίηση. Πρακτικά όμως θέλει ιδιαίτερη προσοχή στον διαχωρισμό των preallоction μεταξύ κόμβων, γιατί ο κώδικας του ext4 έχει δομηθεί με τρόπο που τον κάνει αρκετά εύρωστο ως προς την αναγνώριση περιπτώσεων παραβίασης της λογικής λειτουργίας του.



Σχήμα 6.6: Διάγραμμα ροής της συνάρτησης `ext4_mb_new_blocks` μετά τις τροποποιήσεις

Για παράδειγμα, αν επιλέξουμε να αγνοήσουμε κάποιο διαθέσιμο per-inode

preallocation στην συνάρτηση ext4_mb_use_preallocated, απλά και μόνο επειδή αυτό δεν βρίσκεται στον επιθυμητό NUMA κόμβο, τότε με μεγάλη πιθανότητα θα είναι ανεπιτυχής η ext4_mb_use_preallocated. Μετά θα προετοιμαστούμε για την συνάρτηση ext4_mb_regular_allocator καλώντας πρώτα την ext4_mb_normalize_request. Στην τελευταία συνάρτηση όμως γίνεται εκ νέου έλεγχος μέσω του BUG_ON macro ότι δεν υπάρχει κάποιο preallocation του συγκεκριμένου inode που καλύπτει ήδη το ζητούμενο εύρος λογικών block. Επομένως, αγνοώντας κάποιο κατάλληλο ανά-inode preallocation στην συνάρτηση ext4_mb_use_preallocated, θα οδηγηθούμε με βεβαιότητα σε σφάλμα στην συνέχεια της εκτέλεσης.

Για να αποφύγουμε αυτό το πρόβλημα, φροντίζουμε να έχουμε ξεχωριστές λίστες ανά NUMA κόμβο για τα per-inode preallocations, και ο επιθυμητός κόμβος να αποτελεί παράμετρο των συναρτήσεων που αναφέρθηκαν στην αμέσως προηγούμενη παράγραφο.

Για τα preallocation του locality group δεν παρουσιάζεται το ίδιο θέμα, καθώς μπορούμε απλά να αγνοήσουμε τα σχετικά preallocation άλλων κόμβων. Βέβαια αυτή η πρακτική θεωρητικά μπορεί να οδηγήσει στην αχρείαστη προσπέλαση preallocation που αφορούν άλλους κόμβους και τα οποία δεν μπορούμε να αξιοποιήσουμε. Υπενθυμίζεται όμως ότι η λίστες των locality group preallocation είναι ανά επεξεργαστικό πυρήνα. Συνεπώς, αν προσέχουμε ώστε οι δεσμεύσεις μέσω της ext4_mb_regular_allocator να σέβονται την κατά NUMA τοπικότητα, τότε δεν μπορούν να προκύψουν preallocation που αφορούν ξένους NUMA κόμβους. Ακόμα και αν πραγματοποιηθεί δέσμευση σε ξένο κόμβο από σφάλμα στις τροποποιήσεις που έγιναν, το πιθανό παραγόμενο preallocation έχει προοπτική να διαγραφεί αν κάποια στιγμή δεν επαρκεί ο χώρος σε μελλοντική δέσμευση. Επομένως, δεν είναι απαραίτητο ότι αν προκύψουν στάσιμα preallocation, αυτά θα εμποδίσουν την καλύτερη αξιοποίηση του χώρου επί του μέσου.

Το πιο σύνθετο μέρος των τροποποιήσεων που έγιναν στην συνάρτηση ext4_mb_new_blocks αποτέλεσε η διατήρηση της λογικής στην διαχείριση των σφαλμάτων που μπορούν να προκύψουν από την συνάρτηση ext4_mb_regular_allocator. Όπως θα φανεί και σε επόμενο διάγραμμα ροής, η συνάρτηση ext4_mb_regular_allocator έχει δύο είδη σφαλμάτων:

- **Σφάλματα χαμηλής προτεραιότητας:** Μπορούν να προκύψουν μέσω κλήσης της ext4_mb_good_group_nolock για ένα συγκεκριμένο group, αλλά δεν είναι αρκετά σοβαρά για να μας αποτρέψουν από την συνέχιση της αναζήτησης για άλλα υποψήφια group. Αν βρεθεί κάποιο group στο τέλος της ext4_mb_regular_allocator αγνοείται το χαμηλής προτεραιότητας σφάλμα, διαφορετικά επιστρέφεται στην καλούσα συνάρτηση.

- **Σφάλματα υψηλής προτεραιότητας:** Σχετίζονται με δυσλειτουργία στον buddy allocator. Αν προκύψει, διακόπτουμε άμεσα την αναζήτηση και επιστρέφουμε το σφάλμα στην καλούσα συνάρτηση.

Για την συνάρτηση `ext4_mb_new_blocks` στην αρχική της μορφή όμως δεν υπάρχει διαφοροποίηση μεταξύ των σφαλμάτων, και αν λάβει σφάλμα από την `ext4_mb_regular_allocator`, το αναφέρει άμεσα. Εμείς από την άλλη αλλάζουμε την σειρά κάποιων λειτουργιών της `ext4_mb_new_blocks`, καθώς έχουμε τον μετασχηματισμό που φαίνεται στο σχήμα 6.7. Ο επιπλέον κώδικας λόγω του μετασχηματισμού έχει σημειωθεί με μπλε χρώμα. Αναφέρεται ότι έχουν γίνει πολλές και σημαντικές απλοποιήσεις για την λειτουργία της `ext4_mb_new_blocks`. Στον ψευδοκώδικα παρέχεται τόση πληροφορία όση χρειάζεται για να φανεί η ανάγκη για πιο λεπτομερή διαχείριση σφαλμάτων εντός της συνάρτησης αυτής.

Πλέον λοιπόν δεν μπορούμε απλά να μεταχειριζόμαστε με τον ίδιο τρόπο τα χαμηλής και υψηλής προτεραιότητας σφάλματα, και χρειάζεται να εισάγουμε και στην `ext4_mb_new_blocks` την λογική αυτής της διαφοροποίησης που εμπεριέχεται αρχικά μόνο στην `ext4_mb_regular_allocator`. Αντιλαμβανόμαστε ότι μπορούμε να αγνοήσουμε προσωρινά ένα χαμηλής προτεραιότητας σφάλμα μέχρι την ολοκλήρωση της αναζήτησης για όλους τους NUMA κόμβους. Το αναφέρουμε λοιπόν μόνο αφότου η αναζήτηση αποτύχει σε όλους τους κόμβους.

Ιδιαίτερη προσοχή χρειάζεται ώστε να μην παραβιάσουμε μια ορισμένη ιδιότητα της συνάρτησης `ext4_mb_new_blocks`, η οποία δεν είναι εμφανής στον ψευδοκώδικα λόγω των απλοποιήσεων που έγιναν. Αναλυτικότερα, στην αρχική της εκδοχή επιτρέπεται να ξαναδοκιμάσουμε από το σημείο της κλήσης της συνάρτησης `ext4_mb_regular_allocator` μόνο αν μπορούν να διαγραφούν αρκετά preallocation ώστε αυτό να έχει νόημα (`ext4_mb_discard_preallocations_should_retry`). Δεν θα μπορούσαμε όμως να φτάσουμε σε αυτόν τον έλεγχο αν είχε προηγηθεί χαμηλής προτεραιότητας σφάλμα, καθώς τότε θα το είχαμε αναφέρει άμεσα. Έτσι, και στην τροποποιημένη εκδοχή της `ext4_mb_new_blocks` σταματάμε να κάνουμε αυτόν τον έλεγχο από την στιγμή που θα καταγραφεί χαμηλής προτεραιότητας σφάλμα για οποιονδήποτε κόμβο. Αλλιώς γίνεται η υπόθεση ότι μπορεί να προκύψει απροσδιόριστη συμπεριφορά από την διπλή αναζήτηση στον ίδιο κόμβο ενόσω εκκρεμεί το σφάλμα.

Στον κώδικα που συνοδεύει την παρούσα εργασία έχουν παρατεθεί σχόλια που εξηγούν αναλυτικότερα τις επιμέρους ιδιοτροπίες των αλλαγών που έγιναν στην συνάρτηση `ext4_mb_new_blocks`.

ext4_mb_new_blocks: Ψευδοκώδικας της αρχικής εκδοχής

```
SEARCH_PREALLOCATED(request)
if request → status ≠ FOUND then
    try :
        REGULAR_ALLOCATOR(request)
    catch Error :
        return Error
if request → status = FOUND then
    return request → newblock
else
    return −ENOSPC
```

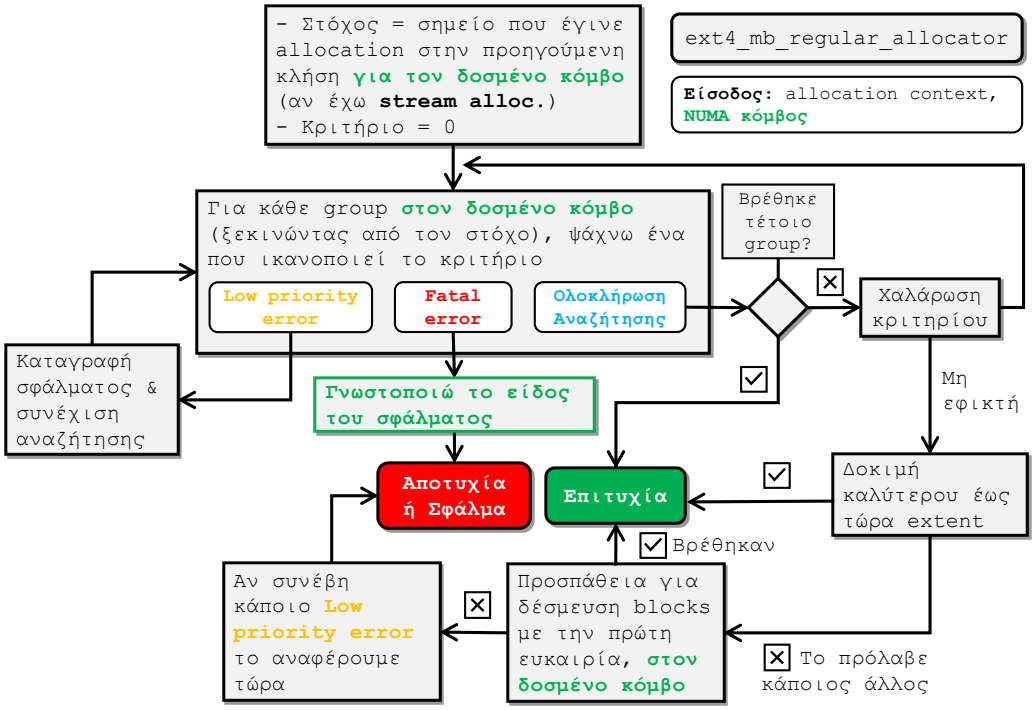
ext4_mb_new_blocks: Οι αλλαγές μετά τον μετασχηματισμό

```
for node n ∈ NUMA nodes do
    SEARCH_PREALLOCATED(request)
    if request → status ≠ FOUND then
        try :
            REGULAR_ALLOCATOR(request)
        catch Error :
            if Error is critical then
                return Error
            else
                record Error
    if request → status = FOUND then
        return request → newblock
if Error recorded then
    return Error
else
    return −ENOSPC
```

Σχήμα 6.7: Ο μετασχηματισμός στην συνάρτηση `ext4_mb_new_blocks`

Συνάρτηση ext4_mb_regular_allocator

Έχοντας ήδη αναφέρει αρκετές λεπτομέρειες της υλοποίησης της, παρουσιάζουμε και τις αλλαγές που έγιναν στην συνάρτηση `ext4_mb_regular_allocator`, με την οποία κλείνουμε και την ενότητα των αλλαγών που έγιναν στην λογική του multiblock allocation του ext4. Στον πίνακα 6.1 αναλύονται και τα κριτήρια που αναφέρονται στα σχήματα.



Σχήμα 6.8: Διάγραμμα ροής της συνάρτησης `ext4_mb_regular_allocator` (οι τροποποιήσεις φαίνονται με πράσινο χρώμα)

Κριτήριο	Συνθήκη για το group (ή το αίτημα)
0	Όπως στο κριτήριο 1, αλλά το μήκος του αιτήματος είναι δύναμη του 2
1	Υπάρχει fragment (εύρος συνεχόμενων block) με μήκος ≥ Μήκος αιτήματος
2	Συνολικά διαθέσιμα block ≥ Μήκος αιτήματος
3	Καμία συνθήκη

Πίνακας 6.1: Επεξήγηση των κριτηρίων που χρησιμοποιεί η συνάρτηση `ext4_mb_regular_allocator`

Αν και το διάγραμμα ροής φαίνεται περίπλοκο, στην πραγματικότητα αυτό σχετίζεται περισσότερο με την εγγενή πολυπλοκότητα της συνάρτησης. Οι τροποποιήσεις που χρειάζεται να πραγματοποιήσουμε στην συνάρτηση αυτή είναι αρκετά απλές:

- Αν έχουμε stream allocation για το τρέχον αίτημα, διαλέγουμε τον στόχο που αναλογεί στον NUMA κόμβο που έχει δοθεί ως παράμετρος.

- Περιορίζουμε την εμβέλεια της αναζήτησης μας στον NUMA κόμβο που έχει δοθεί ως παράμετρος.
- Αν προκύψει σφάλμα υψηλής προτεραιότητας, το γνωστοποιούμε στην καλούσα συνάρτηση μέσω της νεοεισαχθείσας παραμέτρου `fatal_err`.

6.3 NUMA IO Accounting

Θέλουμε ιδανικά ο πυρήνας να παρέχει μια διεπαφή προς τους χρήστες, μέσω της οποίας θα γίνονται διαθέσιμα στατιστικά Εισόδου/Εξόδου (I/O) ανά διεργασία και ανά NUMA κόμβο για τις επιμέρους διεργασίες. Αυτή μπορεί να εξυπηρετήσει κατά κύριο λόγο το userspace component, εφόσον βασικό μέρος της λογικής του περί λήψης αποφάσεων αποτελεί η εύρεση του κόμβου στον οποίο κάθε διεργασία πραγματοποιεί τις περισσότερες προσβάσεις. Περισσότερο όμως, η διεπαφή αυτή μπορεί να αποτελέσει εύχρηστο εργαλείο για την αποσφαλμάτωση κατά την επέκταση του ext4: χαρακτηριστικά αναφέρεται ότι πολλές διορθώσεις στον κώδικα έγιναν παρατηρώντας περιοδικά μέσω της εν λόγω διεπαφής το ποσοστό προσβάσεων σε μη επιθυμητό NUMA κόμβο εκ μέρους νημάτων του filebench.

Αυτή η διεπαφή μοιάζει πολύ με το αρχείο `io` του συστήματος αρχείων ειδικού σκοπού proc. Τα προβλήματα με το αρχείο αυτό είναι 1) ότι δεν καταγράφει στατιστικά I/O για την λειτουργία DAX και 2) ότι η καταγραφή που πραγματοποιεί δεν γίνεται σε επίπεδο NUMA κόμβων, δηλαδή οι πληροφορίες που παρέχει είναι αθροιστικές για όλους τους κόμβους. Επομένως, βασιζόμενοι στην υποδομή του αρχείου `io` του procfs, σκοπός μας είναι να προσθέσουμε ένα αρχείο `numa_io` που θα διορθώνει τις ελλείψεις που αναφέρθηκαν. Ο λόγος για τον οποίο δημιουργούμε καινούργιο αρχείο και δεν επεκτείνουμε το ήδη υπάρχον, άπτεται κυρίως στην διατήρηση συμβατότητας με εγκατεστημένα προγράμματα του συστήματος των οποίων η ορθή λειτουργία ενδεχομένως να εξαρτάται από την συγκεκριμένη δομή του αρχείου `io`.

Στην συνέχεια εξετάζουμε τα βασικά βήματα της υλοποίησης του καινούργιου αυτού αρχείου.

6.3.1 Καταχώρηση του νέου αρχείου στο procfs

Πρώτο μας βήμα είναι να ελέγξουμε την υλοποίηση του αρχείου `io` στον πηγαίο κώδικα του procfs. Έτσι καταφέρνουμε από την μια μεριά να προσδιορίσουμε από ποια δομή της αντίστοιχης διεργασίας αντλούνται τα σχετικά στατιστικά, ώστε να την επεκτείνουμε κατάλληλα. Από την άλλη, πληροφορούμαστε για την μεθοδολογία εισαγωγής καινούργιων αρχείων στο procfs.

Μετά από σχετική αναζήτηση, οδηγούμαστε στο αρχείο `fs/proc/base.c` του πυρήνα. Στον κώδικα του υπάρχουν δύο δομές με καθολική εμβέλεια:

- `struct pid_entry tgid_base_stuff`: Εδώ εμπεριέχονται καταχωρήσεις για τα αρχεία του procfs που υπάγονται στον κατάλογο `/proc/<pid>/task/<tid>/`
- `struct pid_entry tgid_base_stuff`: Αυτή η δομή εμπεριέχει τις αντίστοιχες καταχωρήσεις του καταλόγου `/proc/<pid>/`

Στο παράδειγμα του αρχείου `io`, η υλοποίηση του για κάθε κατάλογο διαφέρει, όπως φαίνεται από τις ακόλουθες γραμμές του αρχείου `fs/proc/base.c`:

```
// static const struct pid_entry tgid_base_stuff[] = { ...
ONE("io", S_IRUSR, proc_tgid_io_accounting),
// static const struct pid_entry tid_base_stuff[] = { ...
ONE("io", S_IRUSR, proc_tid_io_accounting),
```

Διευκρινίζουμε προς την κατανόηση της σημασιολογίας αυτών των συναρτήσεων ότι κατά σύμβαση θεωρούμε πως κάθε διεργασία <pid> συσχετίζεται με τα θυγατρικά της νήματα <tid>. Μεταξύ αυτών υπάρχει και το βασικό νήμα της διεργασίας, που λαμβάνει <tid> ίσο με <pid>. Στην πραγματικότητα όμως, αναφέρεται ως παρένθεση ότι στην υλοποίηση του πυρήνα υπάρχει μόνο η έννοια της διεργασίας (process), η οποία διαθέτει ένα μοναδικό PID (Process ID) και ένα TGID (Thread Group ID). Η οντότητα που θεωρήσαμε ως διεργασία με αναγνωριστικό <pid> έχει TGID = PID, ενώ τα θυγατρικά νήματα έχουν το καθένα δικό του PID, αλλά μοιράζονται το TGID με τον γονέα τους.

Στην περίπτωση του αρχείου `/proc/<pid>/task/<tid>/io` διατίθενται τα στατιστικά εισόδου-εξόδου αποκλειστικά ενός νήματος με αναγνωριστικό <tid> και γονέα <pid> (μπορεί να ισχύει και <pid> = <tid>). Στην περίπτωση του αρχείου `/proc/<pid>/io` έχουμε συσσωρευτικά στατιστικά εισόδου-εξόδου για όλα τα νήματα που υπάγονται στον γονέα <pid> (όπως είπαμε, αυτό συμπεριλαμβάνει και το βασικό νήμα της διεργασίας που λαμβάνει <tid> = <pid>). Αν για παράδειγμα επιθυμούσαμε να λάβουμε τα στατιστικά της ίδιας της διεργασίας και όχι των παιδιών της, θα εξετάζαμε το αρχείο `/proc/<pid>/task/<pid>/`.

Από τα παραπάνω γίνεται άμεσα αντιληπτό το πως θα προσθέσουμε τα καινούργια αρχεία `/proc/<pid>/task/<tid>/numa_io` και `/proc/<pid>/numa_io`, και τι χρειάζεται να προσέξουμε στην σημασιολογία τους για να συνάδει με την λογική του αρχείου `io`.

6.3.2 Προσθήκη Πεδίων NUMA I/O Accounting

Εξετάζοντας τις υλοποιήσεις των συναρτήσεων `proc_tgid_io_accounting` και `proc_tid_io_accounting` σύντομα καταλήγουμε στην δομή `struct task_io_accounting` μέσω των πεδίων `read_bytes` και `write_bytes`. Οδηγούμαστε στο αρχείο `/include/linux/task_io_accounting.h` του πηγαίου κώδικα του πυρήνα, στο οποίο πραγματοποιούμε τις επόμενες προσθήκες:

```
// struct task_io_accounting { ...
u64 numa_read_bytes[2];
u64 numa_write_bytes[2];
// ...
```

Για κάθε είδος πρόσβασης (ανάγνωση ή εγγραφή) έχουμε από ένα πεδίο που αποτελεί συστοιχία τιμών, με κάθε τιμή να αντιστοιχεί σε έναν NUMA κόμβο. Ιδανικά, θα θέλαμε μια ενιαία παράμετρο μεταξύ της υλοποίησης του ext4 και των αρχείων του καταλόγου `/include/linux/`, η οποία θα αναπαριστούσε τον αριθμό των NUMA κόμβων του συστήματος (ρόλος παρόμοιος της σταθεράς `EXT4_NUMA_NUM_NODES`, αλλά ευρύτερος). Κατά την συγγραφή της εργασίας, δεν ήταν άμεσα αντιληπτό το πως αυτό είναι εφικτό με κομψό τρόπο ή σε συμφωνία με την υποδομή που παρέχει το σύστημα μεταγλώττισης του πυρήνα για τέτοιες περιπτώσεις. Για παράδειγμα, θα ήταν προτιμότερο να ορίσουμε προς αυτήν την κατεύθυνση μια σχετική παράμετρο που ρυθμίζεται μέσω της εντολής `make menuconfig`. Ωστόσο, τα χρονικά περιθώρια συγγραφής του κώδικα δεν επέτρεπαν την διερεύνηση αυτού του ενδεχομένου, άρα και επιλέχθηκε εν προκειμένω ο αριθμός των NUMA κόμβων του συστήματος να παραμείνει hard-coded τιμή.

6.3.3 Συναρτήσεις ενημέρωσης των νέων πεδίων

Επόμενο βήμα μας είναι να προσδιορίσουμε σε ποια σημεία του πυρήνα ενημερώνονται τα πεδία `read_bytes` και `write_bytes` της δομής `task_io_accounting`. Ύστερα από αναζήτηση, βρίσκουμε τις συναρτήσεις `task_io_account_read`, `task_io_account_write` και `task_io_accounting_add`. Για διευκόλυνση στην ανάγνωση της υλοποίησης, προσθέτουμε τα άμεσα ανάλογα των συναρτήσεων αυτών, δηλαδή τις συναρτήσεις με πρόθεμα `task_numa_io_account_` του πηγαίου κώδικα που συνοδεύει την παρούσα εργασία.

Καταλαβαίνουμε ότι οι κλήσεις των συναρτήσεων με πρόθεμα `task_io_account_` υποδεικνύουν που χρειάζεται δυνητικά να προσθέσουμε τις καινούργιες συναρτήσεις με πρόθεμα `task_numa_io_account_`. Ύστερα από εξέταση μέσω του `grep`, και αγνοώντας περιπτώσεις συστημάτων αρχείων που πραγ-

ματοποιούν απευθείας το δικό τους I/O accounting, καταλήγουμε στις συναρτήσεις του ακόλουθου πίνακα.

Κατηγορία	**Αρχείο**	**Συνάρτηση**
Memory Management	`/mm/readahead.c`	`read_cache_pages`
	`/mm/page-writeback.c`	`account_page_dirtied`
Block Device	`/block/blk-core.c`	`submit_bio`
	`/fs/block_dev.c`	`__blkdev_direct_IO`
		`__blkdev_direct_IO_simple`
Direct-IO	`/fs/direct-io.c`	`submit_page_section`
	`/fs/iomap/direct-io.c`	`iomap_dio_bio_actor`

Πίνακας 6.2: Σημεία στον πυρήνα Linux στα οποία πραγματοποιείται I/O accounting

Από τις επιμέρους κατηγορίες θα εστιάσουμε στην υλοποίηση κυρίως στις "Memory Management" και "Block Device," αφού η "Direct-IO" δεν παρουσιάζει ιδιαίτερο ενδιαφέρον για αυτήν την εργασία. Ωστόσο, ο τρόπος συμπερίληψης της ελπίζουμε ότι θα γίνει εμφανής μέχρι το τέλος της ενότητας. Παρατηρούμε ότι καμία κατηγορία δεν φαίνεται να παραπέμπει στην λειτουργία DAX. Όπως αναφέρθηκε, το I/O accounting δεν υποστηρίζεται καθόλου για την λειτουργία αυτή στην αρχική εκδοχή του πυρήνα. Επομένως, χρειάζεται πρώτα να εντοπίσουμε το κατάλληλο σημείο εισαγωγής των καινούργιων συναρτήσεων καταγραφής. Παρατηρώντας την αλληλουχία συναρτήσεων του πυρήνα που σχετίζονται με τις προσβάσεις τύπου DAX, εντοπίζουμε την υποψήφια συνάρτηση `dax_iomap_actor` του αρχείου `fs/dax.c`.

Οι συναρτήσεις με πρόθεμα `task_numa_io_account_` χρειάζονται όμως ως παράμετρο το αναγνωριστικό του NUMA κόμβου του οποίου τα στατιστικά θέλουμε να ενημερώσουμε. Επομένως, σε κάθε συνάρτηση που μας ενδιαφέρει να καταγράψουμε στατιστικά I/O οφείλουμε να διαθέτουμε έναν τρόπο ανάκτησης του σωστού NUMA κόμβου βάσει των δοσμένων παραμέτρων. Αυτή η διαδικασία αποδεικνύεται μη τετριμμένη, και για αυτό παρουσιάζουμε τις επιμέρους περιπτώσεις στην συνέχεια.

Ανάκτηση NUMA κόμβου στον κώδικα για την λειτουργία DAX

Στην συνάρτηση `dax_iomap_actor`, στην οποία θέλουμε να κάνουμε καταγραφή I/O στατιστικών, δίνεται ως παράμετρος ένα αντικείμενο τύπου `dax_device`, και εντός της δομής επανάληψης υπολογίζεται το ανάλογο `sector` επί

της συσκευής. Επιθυμούμε λοιπόν να δημιουργήσουμε μια συνάρτηση ονόματι `get_numa_node` που θα συνδυάζει αυτές τις δύο πληροφορίες για να μας υποδείξει τον αντίστοιχο NUMA κόμβο.

Εφόσον η συνάρτηση `get_numa_node` εξαρτάται από τον driver που αναλογεί στο αντικείμενο τύπου `dax_device`, είναι λογικό να την ορίσουμε ως μέλος της αφαιρετικής δομής `struct dax_operations`. Η περιγραφή της δομής αυτής βρίσκεται στο αρχείο `include/linux/dax.h`. Γενικά θα δούμε ότι πολλές αφαιρετικές διεπαφές τείνουν να βρίσκονται στον κατάλογο `include/linux`. Με αυτόν τον τρόπο, κατασκευάσαμε την διεπαφή την οποία θα καλεί το αρχείο `dax.c` και θα υλοποιήσει κατάλληλα ο driver του device mapper.

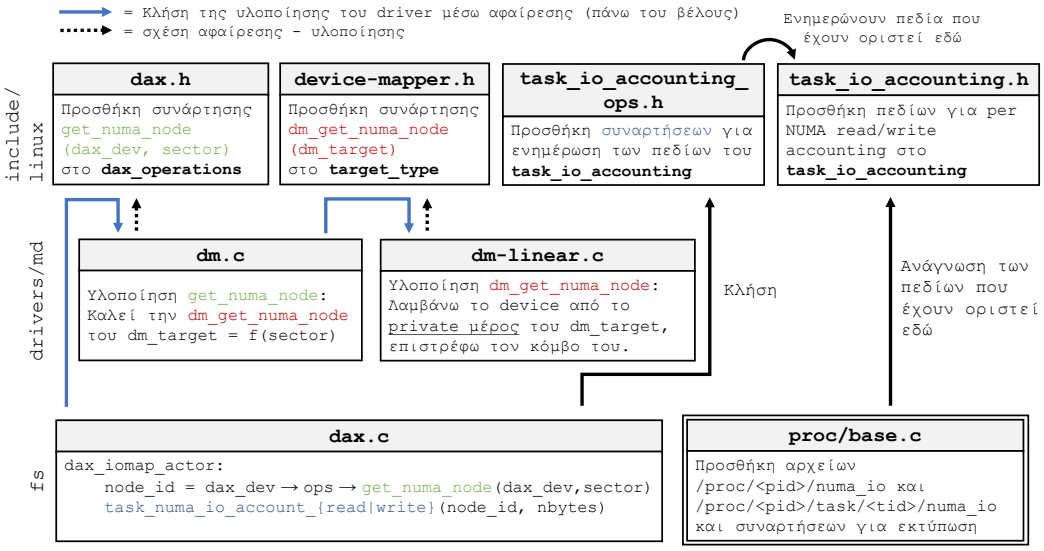
Επόμενο μας βήμα είναι η υλοποίηση της συνάρτησης `get_numa_node` στον driver που μας ενδιαφέρει. Δεδομένου ενός αιτήματος προς μια συσκευή του device mapper, η γενική λογική λειτουργίας του οδηγού είναι ο αρχικός προσδιορισμός της συσκευής-στόχου (αντικείμενο τύπου `dm_target`) που αναλογεί στο δοθέν offset του mapped device, και ύστερα η ικανοποίηση του αιτήματος επί του στόχου που βρέθηκε, με τρόπο που εξαρτάται από το είδος του mapped device (linear, striped, κλπ). Για να γίνει πιο κατανοητή η λογική που περιγράφτηκε, στην συνέχεια παρουσιάζεται το παράδειγμα της συνάρτησης `dm_dax_copy_from_iter` του αρχείου `drivers/md/dm.c` του πηγαίου κώδικα του πυρήνα.

```
static size_t dm_dax_copy_from_iter(struct dax_device
    ↪ *dax_dev, pgoff_t pgoff, void *addr, size_t bytes,
    ↪ struct iov_iter *i)
{
        struct mapped_device *md = dax_get_private(dax_dev);
        sector_t sector = pgoff * PAGE_SECTORS;
        struct dm_target *ti;
        long ret = 0;
        int srcu_idx;

        ti = dm_dax_get_live_target(md, sector, &srcu_idx);

        if (!ti)
                goto out;
        if (!ti->type->dax_copy_from_iter) {
                ret = copy_from_iter(addr, bytes, i);
                goto out;
        }
        ret = ti->type->dax_copy_from_iter(ti, pgoff, addr,
            ↪ bytes, i);
 out:
        dm_put_live_table(md, srcu_idx);

        return ret;
}
```

Την δουλειά προσδιορισμού του στόχου αναλαμβάνουν οι κλήσεις στις συναρτήσεις dm_dax_get_live_target και dm_put_live_table. Μεταξύ αυτών, η ικανοποίηση του αιτήματος στο συγκεκριμένο είδος στόχου γίνεται μέσω του πεδίου type τύπου target_type του αντικειμένου dm_target. Καταλαβαίνουμε λοιπόν ότι πρέπει πρώτα να ορίσουμε μια συνάρτηση get_numa_node στο επίπεδο αφαίρεσης που προσφέρει η δομή target_type που περιγράφεται στο αρχείο include/linux/device-mapper.h. Ύστερα, υλοποιούμε την συνάρτηση αυτή στο είδος συσκευής που μας ενδιαφέρει, κοινώς την γραμμική συσκευή η οποία έχει την λογική υλοποίησης της στο αρχείο drivers/md/dm-linear.c.

Έχοντας κάνει τα παραπάνω, η υλοποίηση της επιθυμητής συνάρτησης dm_dax_get_numa_node στο αρχείο drivers/md/dm.c είναι εντελώς ανάλογη του παραδείγματος της συνάρτησης dm_dax_copy_from_iter. Αναθέτουμε τον δείκτη της στο αντικείμενο dm_dax_ops και πλέον μπορεί να κληθεί κανονικά από το αρχείο fs/dax.c.

Σημειώνεται ότι η υλοποίηση της διεπαφής get_numa_node για αντικείμενα τύπου block_device είναι σχεδόν πανομοιότυπη. Αρκεί να την ορίσουμε την διεπαφή αυτήν στην περιγραφή της δομής block_device_operations του αρχείου include/linux/blkdev.h, και να την υλοποιήσουμε στο αρχείο drivers/md/dm.c. Η υλοποίηση έχει τόσα πολλά κοινά με αυτήν του dax_operations, που έχουμε ορίσει την γενικότερη συνάρτηση dm_general_get_numa_node την οποία καλούν οι συναρτήσεις dm_blk_get_numa_node και dm_dax_get_numa_node.



Σχήμα 6.9: Ο τρόπος υλοποίησης της καταγραφής στατιστικών I/O ανά NUMA κόμβο για την λειτουργία DAX

Στο σχήμα 6.9 φαίνονται όλα όσα έχουμε αναφέρει έως τώρα, χωρίς όμως να περιλαμβάνεται η επέκταση για υποστήριξη της διεπαφής get_numa_node αντικειμένων τύπου block_device.

Ανάκτηση NUMA κόμβου στον κώδικα της κατηγορίας "Block device"

Στις συναρτήσεις submit_bio, __blkdev_direct_IO_simple και __blkdev_direct_IO, είτε δίνεται ως παράμετρος ένα αντικείμενο τύπου bio, είτε κατασκευάζεται εντός αυτής για τους σκοπούς της συνάρτησης. Η δομή bio κάνει άμεσα διαθέσιμο το sector στο οποίο γίνεται η πρόσβαση, οπότε αρκεί απλά να χρησιμοποιήσουμε την διεπαφή get_numa_node που προσφέρει η δομή block_device. Τον τρόπο δημιουργίας της διεπαφής περιγράψαμε αμέσως πριν.

Ανάκτηση NUMA κόμβου στον κώδικα του Memory Management

Σε αμφότερες τις συναρτήσεις read_cache_pages και account_page_dirtied δίνεται ως παράμετρος ένα mapping τύπου address_space, και δουλεύουμε με σελίδες (αντικείμενα τύπου struct page). Από το mapping μπορούμε να ανακτήσουμε το σχετικό inode. Αν προσδιορίσουμε μια μέθοδο ανάκτησης του sector από το inode και την σελίδα, μπορούμε αργότερα να αξιοποιήσουμε την διεπαφή get_numa_node της δομής block_device για να επιτύχουμε την εύρεση του NUMA κόμβου.

Σημειώνεται ότι η λογική αντιστοίχισης $(\texttt{inode}, \texttt{page}) \rightarrow \texttt{sector}$ εμπεριέχεται στο σύστημα αρχείων που χρησιμοποιούμε. Επομένως, φτιάχνουμε μια διεπαφή page_sector στον πυρήνα, η οποία αναλαμβάνει να κάνει την αντιστοίχιση, και μπορεί να υλοποιηθεί ξεχωριστά σε κάθε σύστημα αρχείων.

Κατάλληλο μέρος για τον ορισμό της διεπαφής αποτελεί η δομή inode_operations του αρχείου include/linux/fs.h. Υλοποιούμε την διεπαφή στο ext4 μέσω της συνάρτησης ext4_page_sector που ορίζουμε στο αρχείο fs/ext4/inode.c, με λογική βασισμένη σε μέρος της προϋπάρχουσας συνάρτησης ext4_mpage_readpages. Τέλος, αναθέτουμε τον δείκτη της συνάρτησης υλοποίησης στο καινούργιο πεδίο page_sector της δομής ext4_file_inode_operations του αρχείου fs/ext4/file.c.

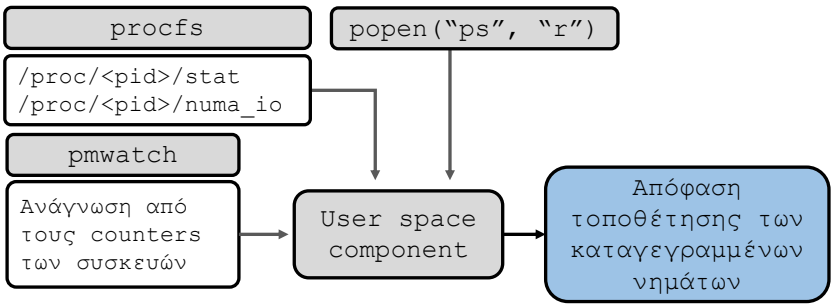
Με όλη την δουλειά που περιγράφηκε σε αυτήν την ενότητα, πλέον έχουμε διαθέσιμα τα στατιστικά που είχαμε σκοπό να κάνουμε διαθέσιμα, μέσω των καινούργιων αρχείων numa_io του procfs. Λείπει πλήρης υποστήριξη για Direct IO, αλλά αυτό είναι κάτι που υλοποιείται εύκολα αν χρησιμοποιήσουμε ως γνώμονα όσα αναφέρθηκαν.

6.4 Userspace Component

Αν και το userspace component δεν έφτασε σε σημείο ανάπτυξης στο οποίο θα μπορούσαν να παρουσιαστούν σχετικά αποτελέσματα στην ενότητα *Αξιολόγηση*, αποτελεί μια αξιόλογη βάση για μελλοντική επέκταση της εργασίας. Ακολούθως θα περιγράψουμε το πως διαμορφώθηκε η σχετική υποδομή του καταλόγου `userspace` του πηγαίου κώδικα της εργασίας, ώστε να καταγραφεί η έως τώρα πρόοδος. Στο τέλος του κεφαλαίου θα εξετάσουμε σχετικά προβλήματα που παρουσιάστηκαν κατά τον σχεδιασμό του userspace component, ώστε να λειτουργήσουν ως υπόδειξη για τον πιθανό τρόπο επέκτασης του.

6.4.1 Σχεδιασμός και Υλοποίηση

Στο σχήμα 6.10 παρουσιάζεται αφαιρετικά ο τρόπος που το userspace component, στην τωρινή του μορφή, συλλέγει δεδομένα σχετικά με την κατάσταση του συστήματος.



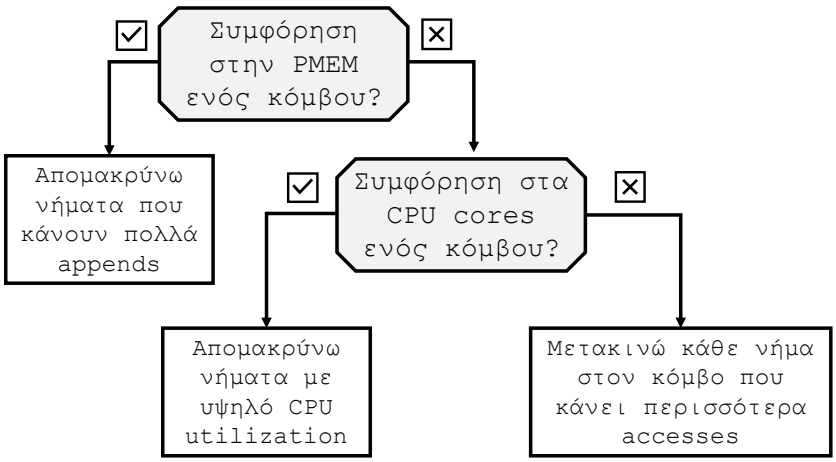
Σχήμα 6.10: Σχεδιασμός του userspace component

Η συλλογή δεδομένων και η λήψη αποφάσεων για την τοποθέτηση γίνεται ανά ένα προκαθορισμένο κβάντο χρόνου, που έχει οριστεί ίσο με 1 δευτερόλεπτο για συμβατότητα με τους περιορισμούς λειτουργίας του PMWatch. Η κατάσταση του συστήματος την στιγμή συγγραφής της εργασίας θεωρούμε ότι αποτελείται από τα εξής συστατικά στοιχεία:

- Η αξιοποίηση κάθε επεξεργαστικού πυρήνα, όπως εξάγεται από τις πληροφορίες του αρχείου `stat` του proc fs
- Πληροφορίες σχετικά με την ρυθμαπόδοση των μη πτητικών συσκευών, που λαμβάνουμε μέσω του προγράμματος PMWatch της intel
- Μια συλλογή νημάτων που προκύπτει από επεξεργασία της εξόδου του προγράμματος `ps`

- Τα ανά NUMA στατιστικά I/O κάθε νήματος, που λαμβάνουμε μέσω του αρχείου `numa_io` της προηγούμενης ενότητας

Μια περίληψη της πολιτικής τοποθέτησης των νημάτων βάσει των πληροφοριών που λαμβάνουμε από το σύστημα είναι αυτή του σχήματος 6.11. Η πολιτική ακολουθεί στενά την αντίστοιχη που περιγράφεται στο [28], αφού θεωρήθηκε καλή βάση πάνω στην οποία θα μπορούσαμε να είχαμε χτίσει την λογική υπό συνθήκης ρύθμισης του χαρακτηριστικού DAX για κάθε αρχείο.



Σχήμα 6.11: Απλοποιημένη εκδοχή της λογικής τοποθέτησης νημάτων στο userspace component

Σημειώνεται ότι από την λογική αυτή απουσιάζει η πολύ σημαντική παράμετρος του διαμοιρασμού δεδομένων (data sharing) μεταξύ νημάτων. Αν δύο ή περισσότερα νήματα γράφουν συχνά σε ένα διαμοιραζόμενο αρχείο, θέλουμε να φροντίσουμε αυτά να βρίσκονται στον ίδιο κόμβο με το αρχείο. Διαφορετικά, θα παραχθεί αξιοσημείωτη κίνηση δεδομένων στο δίκτυο διασύνδεσης των επεξεργαστών, με προφανείς συνέπειες στην επίδοση των προσβάσεων στις μη πτητικές μνήμες. Θα αναλύσουμε περαιτέρω αυτό το θέμα στην υποενότητα με τα βήματα επέκτασης.

Στην συνέχεια επεξηγούμε τις συνθήκες και τις ενέργειες που καταγράφονται στο παρουσιασμένο διάγραμμα ροής, και αναφέρουμε τα σχετικά σημεία υλοποίησης τους στον κώδικα.

Συμφόρηση στις μη Πτητικές Μνήμες

Με την έκφραση "συμφόρηση στην PMEM ενός κόμβου," εννοούμε ότι υπάρχει ένας κόμβος στον οποίον το συνολικό bandwidth (με την έννοια συνολικό εννοούμε ότι είναι αντιπροσωπευτικό και των αναγνώσεων και των εγγραφών) ξεπερνάει ένα ποσοστό της μέγιστης δυνατής τιμής του, και είναι

διπλάσιο από το συνολικό bandwidth οποιουδήποτε άλλου κόμβου. Η υλοποίηση του ελέγχου φαίνεται στην συνάρτηση `nvmm_contention` του αρχείου `userspace/src/nvmm.c`.

Ορίζουμε το συνολικό bandwidth του κόμβου n ως

$$\texttt{total\_BW}(n) = \texttt{read\_BW}(n) \cdot \frac{1}{3} + \texttt{write\_BW}(n)$$

ώστε να πραγματοποιήσουμε κανονικοποίηση δεδομένου του γεγονότος ότι η μέγιστη ρυθμαπόδοση εγγραφής στις μη πτητικές μνήμες είναι περίπου τρεις φορές μικρότερη από την μέγιστη ρυθμαπόδοση ανάγνωσης. Το μέγιστο bandwidth εγγραφής και ανάγνωσης, όπως και το ποσοστό που θέτουμε ως όριο για την συμφόρηση, παρέχονται στο αρχείο `userspace/headers/nvmm.h`.

Έχουμε πρακτικά δύο τρόπους να λάβουμε τα byte ανάγνωσης και εγγραφής σε κάθε κόμβο: στον πρώτο τρόπο αθροίζουμε μεταξύ των μη πτητικών συσκευών του κόμβου τους σχετικούς μετρητές που λαμβάνουμε μέσω του PMWatch API. Στον δεύτερο τρόπο αθροίζουμε μεταξύ όλων των νημάτων τα σχετικά με τον κόμβο πεδία της διεπαφής `numa_io` του procfs. Ο πρώτος τρόπος σε σχέση με τον δεύτερο έχει μεγαλύτερη ακρίβεια στην πληροφορία που αποδίδει, και μπορεί να αναδεικνύει το φαινόμενο "write amplification" [30]. Όμως αυτός δεν παρέχει απαραίτητα το καλύτερο μέσο σύγκρισης για τον σκοπό μετακίνησης των νημάτων, αφού με τον δεύτερο τρόπο έχουμε σαφή εικόνα για το τι κάνουν συγκεκριμένα τα νήματα υπό την διαχείριση του userspace component.

Απομάκρυνση νημάτων που κάνουν πολλές επεκτάσεις

Όταν υπάρχει συμφόρηση στις μη πτητικές μνήμες, έχουμε δύο σχετικά προφανείς τρόπους για να την διορθώσουμε κατανέμοντας τον φόρτο μεταξύ των κόμβων: ο πρώτος είναι να μετακινήσουμε νήματα και σχετικά με αυτά δεδομένα σε υποχρησιμοποιούμενους κόμβους. Ο δεύτερος είναι απλά να μετακινήσουμε νήματα που κάνουν πολλές εγγραφές που οδηγούν σε δεσμεύσεις (τις αποκαλούμενες "επεκτάσεις" ή "appends") σε υποχρησιμοποιούμενους κόμβους ώστε να δημιουργήσουν καινούργια δεδομένα εκεί. Έτσι, μέσω κατανομής των καινούργιων δεδομένων πετύχουμε κατανομή στις συνολικές προσβάσεις.

Ο πρώτος τρόπος ξεφεύγει του σκοπού της εργασίας, αλλά έχει μελετηθεί στην βιβλιογραφία [13]. Κατά συνέπεια, ο δεύτερος τρόπος είναι αυτός που έχει νόημα να αξιοποιηθεί στην περίπτωση μας. Το πρόβλημα είναι ότι δεν έχουμε τρόπο να διακρίνουμε τα καταγεγραμμένα byte εγγραφών σε overwrite (εγγραφή σε ήδη δεσμευμένα block) και append (δέσμευση καινούργιων block). Αυτό θα μπορούσε να γίνει μέσω επέκτασης της δουλείας της ενότητας *NUMA IO Accounting*, αλλά θα απαιτούσε καταγραφή ειδικών στατιστικών εντός του

συστήματος αρχείων. Προς το παρόν αρκούμαστε στην απλουστευτική υπόθεση ότι οι εγγραφές κάθε νήματος είναι κατά σταθερό ποσοστό επεκτάσεις, οπότε χρησιμοποιούμε τα συνολικά byte εγγραφών ως υπόδειξη για τις μετακινήσεις νημάτων.

Επειδή δεν θέλουμε να μετακινήσουμε τόσα νήματα σε έναν κόμβο που θα δημιουργηθεί εκεί καινούργια συμφόρηση στις μη πτητικές του μνήμες, εισάγουμε για κάθε κόμβο την έννοια του πλεονάσματος αξιοποιούμενης ρυθμαπόδοσης (για τον κόμβο που βρίσκεται σε συμφόρηση) και αντίστοιχα του ελλείμματος (για τους υποχρησιμοποιούμενους κόμβους). Αυτές οι ποσότητες, που θα χαρακτηρίζουμε από κοινού ως `node_bytes_mean_centered` του κόμβου n, ορίζονται με φυσικό τρόπο ως εξής:

$$\texttt{node\_bytes\_mean\_centered}(n) = \texttt{total\_bytes}(n) - \overline{\texttt{total\_bytes}}$$

$$\overline{\texttt{total\_bytes}} = \frac{1}{\#(\texttt{NUMA nodes})} \cdot \sum_{i \in \texttt{NUMA nodes}} \texttt{total\_bytes}(i)$$

Οι τιμές `total_bytes` αντιστοιχούν στα byte που προκύπτουν από το κανονικοποιημένο bandwidth σε συγκεκριμένο κόμβο n, για το περασμένο κβάντο χρόνου. Επομένως, έστω ότι σε συνθήκες συμφόρησης σε επίπεδο μη πτητικών μνημών θέλουμε να μετακινήσουμε ένα νήμα από τον υπό πίεση κόμβο σε ένα κόμβο n, το οποίο νήμα στο περασμένο κβάντο χρόνου έγραψε συνολικά b bytes. Θεωρούμε ότι ικανή συνθήκη για να μην δημιουργηθεί άμεσα νέα συμφόρηση, είναι να μετακινήσουμε το νήμα μόνο αν η ποσότητα $\texttt{node\_bytes\_mean\_centered}(n) + b$ δεν είναι θετική.

Στον διαθέσιμο πηγαίο κώδικα η λογική μετακίνησης νημάτων σε συνθήκες συμφόρησης στις μη πτητικές μνήμες εμπεριέχεται σε συναρτήσεις του αρχείου `userspace/src/target.c`. Οι συναρτήσεις αυτές έχουν στο όνομα τους την συμβολοσειρά "nvmm." Η βασική συνάρτηση είναι η `fix_congestion`, η οποία υλοποιεί το διάγραμμα ροής που παρατέθηκε.

Συμφόρηση στους επεξεργαστικούς πυρήνες ενός κόμβου

Η ανάγνωση του αρχείου `/proc/<tid>/task/<tid>/stat` που αφορά τα στατιστικά του νήματος `<tid>` γίνεται μέσω της συνάρτησης `cpu_stat` του αρχείου `userspace/src/cpu.c`. Οι πληροφορίες που κρατάμε είναι οι παράμετροι utime και stime, που αντιστοιχούν στον συνολικό χρόνο που η διεργασία έχει δρομολογηθεί σε user mode και σε kernel mode αντιστοίχως [25]. Οι τιμές αυτές είναι μετρημένες σε clock tick του πυρήνα.

Ο υπολογισμός του ποσοστού αξιοποίησης του συνόλου των επεξεργαστικών πυρήνων του NUMA κόμβου στον οποίον εκτελείται η διεργασία γίνεται μέσω της συνάρτησης `process_cpu_usage` του αρχείου

`userspace/headers/processes.h`. Για ευκολία αναφοράς, ονομάζουμε αυτήν την ποσότητα του κόμβου n ως $\texttt{cpu\_util}(n)$. Η συμφόρηση ενός κόμβου σε επίπεδο επεξεργαστή ορίζεται εντελώς ανάλογα με την περίπτωση των μη πτητικών μνημών: έχουμε συμφόρηση όταν ένας κόμβος n έχει αξιοποίηση του επεξεργαστή του μεγαλύτερη από ένα όριο ($\texttt{cpu\_util}(n) > 90\%$), και σε όλους τους άλλους κόμβους n' έχουμε $\texttt{cpu\_util}(n') < 0.5 \cdot \texttt{cpu\_util}(n)$.

Απομάκρυνση νημάτων με υψηλή αξιοποίηση του επεξεργαστή

Είναι διαισθητικά αντιληπτό ότι ο άστοχος συνωστισμός υπολογιστικά απαιτητικών διεργασιών σε κάποιον NUMA κόμβο μπορεί να οδηγήσει σε υποβάθμιση της επίδοσης του συστήματος. Συνεπώς, είναι σκόπιμο να συμπεριλάβουμε την αξιοποίηση επεξεργαστικής ισχύος κάθε νήματος στην λογική μετακίνησης του στον κατάλληλο κόμβο. Αυτό είναι κάτι που δεν διαφαίνεται στα benchmark, εφόσον αυτά στοχεύουν στην εξέταση του I/O και δεν αποτελεί πρωτεύουσας σημασίας παράμετρος η ένταση του υπολογισμού.

Παρόλο του προφανή ρόλου στην επίδοση της αποδοτικής κατανομής φόρτου μεταξύ των επεξεργαστικών πυρήνων, αναγνωρίζουμε βάσει των αποτελεσμάτων του κεφαλαίου *Πειράματα* ότι η συμφόρηση σε επίπεδο επεξεργαστή καταλήγει να είναι δευτερεύουσας σημασίας σε σχέση με την συμφόρηση σε επίπεδο μη πτητικών μνημών. Σχετικές παρατηρήσεις έχουν καταγραφεί και στην βιβλιογραφία [28]. Αυτό είναι απτό αν αναλογιστούμε τα φαινόμενα άμεσου κορεσμού της ρυθμαπόδοσης εγγραφών, αλλά και ραγδαίου εκφυλισμού της απόδοσης δεδομένου απομακρυσμένων προσβάσεων. Για αυτό από την μια στο λογικό διάγραμμα που παρατέθηκε υπάρχει η σχετική διάταξη μεταξύ των δύο ελέγχων, και από την άλλη στην υλοποίηση της μετακίνησης των νημάτων λόγω συμφόρησης σε επίπεδο επεξεργαστή ελέγχουμε προληπτικά ότι δεν θα προκληθεί μελλοντική συμφόρηση σε επίπεδο μη πτητικών μνημών.

Η λογική επιλογής και απόρριψης κόμβων προορισμού λόγω συμφόρησης σε επίπεδο επεξεργαστή περιέχεται στις συναρτήσεις `cpu_preferred_node`, `cpu_target_is_suitable` και `cpu_decide_target` του αρχείου `userspace/src/target.c`. Οι συναρτήσεις αυτές αξιοποιούνται καταλλήλως στην συνάρτηση `fix_congestion` του ίδιου αρχείου.

Μετακίνηση νήματος στον κόμβο που επιτελεί τις περισσότερες προσβάσεις

Όταν οι συνθήκες είναι ιδανικές, δηλαδή σε κατάσταση έλλειψης συμφόρησης αμφότερα σε επίπεδο μη πτητικών μνημών και επεξεργαστή, επιθυμούμε να μετακινούμε το κάθε νήμα στον κόμβο από και προς τις μη πτητικές συσκευές του οποίου αυτό μεταφέρει συγκριτικά τον μεγαλύτερο όγκο δεδο-

μένων. Με αυτόν τον τρόπο μπορούμε να περιορίσουμε τις απομακρυσμένες προσβάσεις και την σε περιπτώσεις καταστροφική επίδραση τους στην επίδοση. Εδώ διαδραματίζουν σημαντικό ρόλο τα στατιστικά που λαμβάνουμε από την διεπαφή `numa_io` της προηγούμενης ενότητας. Η συλλογή δεδομένων από την διεπαφή αυτή υλοποιείται στην συνάρτηση `numa_iostat` του αρχείου `userspace/src/numa.c`.

Θα θέλαμε να περιορίσουμε το φαινόμενο της άσκοπης και αλλεπάλληλης μετακίνησης ενός δεδομένου νήματος μεταξύ κόμβων (αποδιδόμενο στα αγγλικά ως "thread oscillation"). Για αυτό και πραγματοποιούμε την μετακίνηση του νήματος μόνο αν ο όγκος δεδομένων πρόσβασης σε έναν κόμβο διαφέρει από τους υπόλοιπους κατά έναν προκαθορισμένο παράγοντα. Στην υλοποίηση μας, η λογική επιλογής του ιδανικού κόμβου για ένα νήμα γίνεται στην συνάρτηση `get_target_node` του αρχείου `userspace/src/target.c`.

6.4.2 Περισσότερες Λεπτομέρειες Υλοποίησης

Όλα όσα περιγράφονται σε αυτήν την αναφορά για την υλοποίηση του userspace component αφορούν το commit με πρόθεμα hash `7f5fa7d` του repository με τον κώδικα της εργασίας. Βάση για την υλοποίηση του userspace component αποτέλεσε ο κώδικας που έγινε διαθέσιμος [14] με αφορμή την δημοσίευση της δουλειάς [23]. Συγκεκριμένα, αξιόλογο μέρος του κώδικα για την διατήρηση λίστας των τρεχόντων νημάτων του συστήματος και την ενημέρωση των πληροφοριών τους βασίστηκε στην δουλειά αυτή. Η σχετική λογική διατίθεται στο αρχείο `userspace/src/processes.c`.

Όπως αναφέρθηκε, προς επίλυση της συμφόρησης σε επίπεδο μη πτητικών μνημών θέλουμε να αναγνωρίσουμε ένα σύνολο νημάτων του υπό πίεση κόμβου προς μετακίνηση, τα οποία πραγματοποιούν σε μεγάλο βαθμό εγγραφές και αρκούν για να καλύψουν το έλλειμμα bandwidth των υπολοίπων κόμβων. Για να γίνει αποδοτικά αυτό, αξιοποιήθηκε μια προσαρμοσμένη εκδοχή της μεθόδου quickselect. Η υλοποίηση της βρίσκεται στο αρχείο `userspace/src/quickselect.c`, και μια σχετική ρουτίνα αξιολόγησης της διατίθεται στο αρχείο `userspace/src/testing/quickselect_test.c`. Η λογική της αξιοποιείται μέσω των συναρτήσεων `compute_nvmm_status` και `is_write_intensive` του αρχείου `userspace/src/target.c`.

Αν και η υλοποίηση της quickselect βοηθάει στην διατήρηση γραμμικότητας της υλοποίησης, σημειώνεται ότι το μεγαλύτερο θέμα κλιμακωσιμότητας προκύπτει στην διαχείριση των τρεχόντων νημάτων μέσω λίστας. Αν κάποια στιγμή επεκταθεί ουσιωδώς το userspace component, είναι σημαντικό να χρησιμοποιηθεί κάποια καταλληλότερη δομή δεδομένων, πιθανώς κάποιο vector με έλεγχο ύπαρξης και δεικτοδότηση των πληροφοριών των νημάτων μέσω hashing.

Ένα άλλο ενδιαφέρον ζήτημα είναι η αποδοτική κάλυψη του ελλείμματος bandwidth στην περίπτωση συμφόρησης σε επίπεδο μη πτητικών μνημών για σύστημα με περισσότερους από δύο κόμβους. Συγκεκριμένα, αν θελήσουμε να κάνουμε ισοκατανομή του αξιοποιούμενου bandwidth προσπαθώντας να σεβαστούμε παράλληλα την τοπικότητα στις προσβάσεις των νημάτων, καταλήγουμε σε ένα πρόβλημα που αλγοριθμικά αναδεικνύεται αρκετά δύσκολο. Στην περίπτωση μας έχουμε δύο NUMA κόμβους, οπότε η επιλογή κόμβου προορισμού είναι τετριμμένη. Βέβαια, αν και το πρόβλημα παρουσιάζει ιδιαιτερότητες στην θεωρητική του διατύπωση, πρακτικά το πιο πιθανό είναι να μην έχει νόημα να λυθεί με τον αποδοτικότερο τρόπο, και να υπάρχει κάποια ικανοποιητική ευρετική μέθοδος.

Επιπροσθέτως, σημειώνεται ότι υπάρχουν και άλλα αρχεία στον πηγαίο που αφορούν περισσότερο πιθανές μελλοντικές επεκτάσεις. Ένα τέτοιο παράδειγμα είναι το αρχείο `userspace/src/dax.c` που αφορά την εναλλαγή του χαρακτηριστικού DAX ενός ανοικτού αρχείου. Μια άλλη περίπτωση είναι αυτή του αρχείου `userspace/src/hijack.c` που δημιουργήθηκε ως πιθανή υποδομή για I/O syscall hijacking, με σκοπό κυρίως την εναλλαγή του χαρακτηριστικού DAX ενός αρχείου πριν το πρώτο άνοιγμα του. Όπως θα αναφερθεί στην ενότητα *Πιο λεπτομερής υποστήριξη DAX* του παραρτήματος, έχει εξεταστεί ο τρόπος άμεσης μετάβασης του χαρακτηριστικού DAX σε ανενεργό (χωρίς δηλαδή αναμονή για να μηδενιστούν οι αναφορές σε αυτό από διεργασίες [9]) προς υποστήριξη του προαναφερθέντος σκοπού.

Για την μεταγλώττιση του userspace component απαιτείται αρχικά να συλλέξουμε τις απαραίτητες εξαρτήσεις μέσω εκτέλεσης του αρχείου `userspace/scripts/dependencies.sh` στον κατάλογο αυτού, και ύστερα να εκτελεστεί η εντολή `make processes` στον κατάλογο `userspace/src`. Οι εξαρτήσεις αφορούν κυρίως το PMWatch και το PMDK, προκειμένου να υπάρξει υποστήριξη πιο σύγχρονης εκδοχής της βιβλιοθήκης libpmem2 [24].

6.4.3 Βήματα επέκτασης του υπάρχοντος σχεδιασμού

Τέλος, θα εξετάσουμε κάποιες κατευθύνσεις επέκτασης του userspace component, αναγνωρίζοντας τις υπάρχουσες ελλείψεις και προτείνοντας πιθανές μεθόδους αντιμετώπισης τους.

Διαμόρφωση πλήρους υποδομής αξιολόγησης

Όπως αναφέρθηκε, ο λόγος που δεν θα παρουσιαστούν αποτελέσματα αξιολόγησης για το userspace component είναι το γεγονός ότι στο παρόν στάδιο παρουσιάζει σημαντικές ελλείψεις. Εκτός αυτού, πραγματοποιήθηκαν κάποιοι πειραματισμοί αλλά το διαθέσιμο μέσο αξιολόγησης, που υπήρξε το filebench,

δεν επαρκούσε για να εξεταστεί πλήρως η αποδοτικότητα της λογικής του userspace component. Συγκεκριμένα, τουλάχιστον στον βαθμό επίγνωσης των χαρακτηριστικών του μετροπρογράμματος, το filebench παράγει κυρίως κίνηση I/O και δεν διαθέτει παράμετρο επιτέλεσης υπολογισμών επί των δεδομένων για να εξεταστεί η αντιμετώπιση συμφόρησης σε επίπεδο επεξεργαστή.

Επιπροσθέτως, τα νήματα του Filebench έχουν αναμενόμενο μοτίβο στην συμπεριφορά τους, το οποίο είναι ίδιο μεταξύ τους. Επομένως, δεν υπάρχει η απαραίτητη δυναμικότητα για να ελεγχθεί η συμπεριφορά του userspace component πέρα της απλής αρχικής τοποθέτησης των νημάτων. Τέλος, κάθε νήμα επιλέγει τυχαία αρχεία κατά την εκτέλεση, άρα δεν μπορεί να υπάρξει εύκολα προτίμηση για κάποιο συγκεκριμένο κόμβο. Επομένως, για την ουσιαστική εξέταση του userspace component στην τωρινή του μορφή χρειάζεται εμπεριστατωμένη υποδομή αξιολόγησης, είτε τροποποιώντας κατάλληλα το filebench είτε χρησιμοποιώντας ένα πιο περίπλοκο benchmark, όπως το RocksDB.

Προσθήκη ελέγχου για διαμοιρασμό αρχείων

Η σημαντικότερη έλλειψη του userspace component αυτήν την στιγμή είναι η απουσία έλεγχου διαμοιρασμού δεδομένων (data sharing) μεταξύ νημάτων. Ο λόγος που δεν έχει υλοποιηθεί ήδη ο έλεγχος αυτός είναι η πολυπλοκότητα υλοποίησης λογικής για την ακριβή γνώση των διαμοιραζομένων αρχείων. Ο λόγος είναι ότι απαιτείται η κατάλληλη δομή δεδομένων για την καταγραφή των ανοικτών αρχείων όλων των επιλεγμένων νημάτων, και για τον εντοπισμό των κοινών ανοικτών αρχείων μεταξύ τους. Πιθανώς η δομή αυτή να αντιστοιχεί σε κάποιο hash table, και να απαιτείται συλλογή των ανοικτών αρχείων μέσω του procfs και της συνάρτησης `fstat` για ανάκτηση του inode, ή μέσω syscall hijacking στις συναρτήσεις `open()` και `close()`.

Ακόμα και αν διαθέτουμε την πληροφορία για τα κοινά ανοικτά αρχεία, χρειαζόμαστε ύστερα τον ορισμό μιας έννοιας βαθμού συσχέτισης μεταξύ των νημάτων, που θα καθορίζεται από τον αριθμό των κοινών ανοικτών αρχείων μεταξύ των νημάτων και ίσως την συχνότητα πρόσβασης στα αρχεία αυτά. Βάσει του βαθμού συσχέτισης, απαιτείται προσδιορισμός μεθόδου clustering των νημάτων δεδομένου επιπλέον της ανάγκης αποδοτικής διαχείρισης των πόρων του συστήματος (επεξεργαστές και μη πτητικές μνήμες).

Έλεγχος ευστάθειας

Μια άλλη σημαντική πτυχή για την αποδοτική λειτουργία του userspace component αποτελεί η διασφάλιση της ευστάθειας του. Εν προκειμένω, με τον όρο ευστάθεια εννοούμε τον κατά το δυνατό περιορισμό των φαινομένων τα-

λάντωσης νημάτων. Αυτό απαιτεί προσεκτική ανάλυση για το πως μπορεί η απόφαση μετακίνησης νήματος λόγω ενός είδους συμφόρησης να οδηγήσει σε δημιουργία κάποιου άλλου είδος συμφόρησης στο άμεσο μέλλον, με αποτέλεσμα την άμεση ή την έμμεση μετακίνηση του ίδιου νήματος κοκ. Αυτό αποδεικνύεται μη τετριμμένο σε θεωρητικό επίπεδο, και για τον έλεγχο του απαιτείται και η κατάλληλη υποδομή αξιολόγησης.

Καλύτερος έλεγχος συμφόρησης στις μη πτητικές μνήμες

Όπως αναφέρεται στην ενότητα *Πειράματα*, παρατηρούνται φαινόμενα απότομου κορεσμού ή και εκφυλισμού της ρυθαμπόδοσης προσβάσεων στις μη πτητικές μνήμες. Επομένως, ο έλεγχος που διαθέτουμε αυτήν την στιγμή, στον οποίο κριτήριο συμφόρησης αποτελεί το ποσοστό αξιοποίησης του μέγιστου δυνατού bandwidth, παρουσιάζει προβλήματα. Ένας καλύτερος έλεγχος θα σύγκρινε το παρατηρούμενο bandwidth σε σχέση με το αναμενόμενο βάσει του όγκου δεδομένων που αιτούνται να γράψουν ή να διαβάσουν τα νήματα. Κοινώς, χρειάζεται να ορίσουμε μια μετρική που αντιπροσωπεύει το πόσο χειρότερα εκτελείται το αίτημα ενός νήματος παρουσίας εξωτερικών αιτημάτων σε σχέση με το σενάριο το αίτημα να εκτελούταν μόνο του. Αν είμαστε κάτω από το όριο του bandwidth αλλά ταυτόχρονα διαθέτουμε κακή τιμή της προαναφερθείσας μετρικής για τα νήματα ενός κόμβου, πάλι μπορεί να είναι αναγκαία η μετακίνηση τους.

Η υλοποίηση ενός ελέγχου συμφόρησης όπως αυτός που περιγράφηκε απαιτεί καταγραφή του μεγέθους των προσβάσεων των νημάτων, που είτε μπορεί να υλοποιηθεί μέσω syscall hijacking, είτε πιθανώς μέσω σχετικών πεδίων του PMWatch.

Υποδομή για υπό συνθήκη αξιοποίηση της page cache

Η DRAM αυτή την στιγμή προσφέρει δύο σημαντικά πλεονεκτήματα σε σχέση με τις μη πτητικές μνήμες: από την μια οι απομακρυσμένες προσβάσεις δεν αντιμετωπίζουν τις σχετικές παθογένειες, και από την άλλη προσφέρει πολύ μεγαλύτερο bandwidth. Για αυτό και ο κύριος σκοπός κατασκευής του userspace component υπήρξε η αξιοποίηση του για την κατά συνθήκες συμπερίληψη της page cache προς βελτιστοποίηση της συνολικής επίδοσης του συστήματος. Μια βασική συνθήκη προς αυτήν την κατεύθυνση αποτελεί η ύπαρξη ενός μηχανισμού επίγνωσης στο userspace για το ποια αρχεία διαθέτουν cached δεδομένα και ποια όχι, προκειμένου να ξέρουμε πότε να μην χρησιμοποιούμε την λειτουργία DAX. Αυτό επί του παρόντος φαίνεται δύσκολο εγχείρημα χωρίς overhead που θα αναιρούσε τον σκοπό του μηχανισμού, και μάλλον θέλει διερεύνηση στον πυρήνα.

Κεφάλαιο 7

Αξιολόγηση

Σε αυτήν την ενότητα θα περιγράψουμε την διαδικασία αξιολόγησης των αλλαγών στο ext4. Η αξιολόγηση προφανώς από την μια έπεται της υλοποίησης, ώστε να διαπιστώσουμε αν και υπό ποιες περιπτώσεις λαμβάνουμε τελικά βελτίωση της λειτουργίας του συστήματος αρχείων. Από την άλλη, θα δούμε ποιες μεθόδους αξιολόγησης αξιοποιήσαμε και κατά την διάρκεια της υλοποίησης, προκειμένου να διασφαλίσουμε την ορθότητα και την ευστάθεια των αλλαγών. Με την έννοια ορθότητα, εννοούμε ότι οι αλλαγές που κάναμε ικανοποιούν τελικά την επιθυμητή ιδιότητα της κατά NUMA τοπικότητας στις δεσμεύσεις επί του μέσου αποθήκευσης, χωρίς να αλλοιώνουν την αρχική λογική σχεδιασμού του ext4 με μη επιθυμητό τρόπο. Με την έννοια ευστάθεια, εννοούμε ότι οι αλλαγές δεν παραβιάζουν την ορθότητα σε άλλα σημεία του κώδικα με τρόπο που οδηγεί το λειτουργικό σύστημα σε αδιέξοδο, πανικό, ή το σύστημα αρχείων να καταστρέψει την δομή του.

Το σύστημα είναι ακριβώς αυτό που περιγράφτηκε στην ενότητα *Χαρακτηριστικά του συστήματος αξιολόγησης* του κεφαλαίου *Πειράματα*. Στην συνέχεια θα αναλύσουμε περαιτέρω τις τρεις πτυχές της διαδικασίας αξιολόγησης (ευστάθεια, ορθότητα και αποδοτικότητα).

7.1 Αξιολόγηση Ευστάθειας

Κατά την διάρκεια της ανάπτυξης των αλλαγών στον κώδικα του πυρήνα, αναμενόμενα αποτέλεσε συχνό φαινόμενο η πρόκληση αδιέξοδων και πανικού του λειτουργικού κατά την δοκιμή των τροποποιήσεων στην εικονική μηχανή. Σε πολλές περιπτώσεις τα σφάλματα στις τροποποιήσεις ήταν άμεσα εμφανή είτε κατά την εκκίνηση της εικονικής μηχανής, είτε κατά την εκτέλεση εργασιών με το FIO ή το Filebench. Αυτού του είδους οι περιπτώσεις μπορούν να εντοπιστούν άμεσα ώστε να αποσφαλματωθούν. Στην γενική περίπτωση όμως,

οφείλουμε να εξετάσουμε οριακές περιπτώσεις, όπως για παράδειγμα τι γίνεται όταν το σύστημα αρχείων διαθέτει ελάχιστα διαθέσιμα block, και υπό συνθήκες πολύωρης και εντατικής χρήσης.

Περιπτώσεις όπως αυτές που περιγράφηκαν μπορούμε να θεωρήσουμε ότι υπάγονται στο λεγόμενο φαινόμενο γήρανσης του συστήματος αρχείων. Αυτό έχει ιδιαίτερο ενδιαφέρον, ειδικά στην αξιολόγηση της ικανότητας ενός συστήματος αρχείων να διατηρεί τα χαρακτηριστικά επίδοσης του σε βάθος χρόνου. Για παράδειγμα, η ιδιότητα αυτή της μη εκφυλιστικής γήρανσης υπήρξε βασικός πυλώνας στον σχεδιασμό του συστήματος αρχείων WineFS [18]. Στην σχετική δημοσίευση αξιοποιήθηκε αρκετά ένα εργαλείο που προσομοιώνει την διαδικασία γήρανσης ενός συστήματος αρχείου, το οποίο ονομάζεται Geriatrix [19].

Το Geriatrix είναι ένα σύνθετο εργαλείο που επιτελεί δημιουργία και τυχαίες προσβάσεις σε καταλόγους και αρχεία εντός αυτών, βάσει επιλεγμένων τυχαίων κατανομών για τον χρόνο γήρανσης, τα μεγέθη των αρχείων, και την δομή των καταλόγων. Μεταξύ άλλων, διαθέτει και παράμετρο για τον βαθμό αξιοποίησης του συστήματος αρχείων. Για τους σκοπούς αυτής της εργασίας, δεν αξιοποιούμε το εργαλείο αυτό για την αξιολόγηση της επίδοσης του τροποποιημένου συστήματος αρχείων, αν και αποτελεί ενδιαφέρουσα προοπτική. Περισσότερο, το αξιοποιήσαμε για να ελέγξουμε την ευστάθεια του υπό συνθήκες εντατικής και εξαντλητικής χρήσης. Ο έλεγχος έγινε με κλήση του Geriatrix ως εξής:

```
$ cd geriatrix/build/
$ ./geriatrix -n 8087252992 -u 0.9 -r 42 -m /mnt/pmem \
       -a ../profiles/agrawal/age_distribution.txt \
       -s ../profiles/agrawal/size_distribution.txt \
       -d ../profiles/agrawal/dir_distribution.txt \
       -x /tmp/age.out -y /tmp/size.out -z /tmp/dir.out \
       -t 1 -i 50 -f 0 -p 0 -c 0 -q 1 -w 2880 -b posix
```

Ο κατάλογος `/mnt/pmem` υπήρξε σταθερό σημείο πρόσδεσης της συσκευής των μη πτητικών μνημών. Εν προκειμένω αξιοποιήσαμε το 90% του διαθέσιμου χώρου. Αφότου βεβαιωνόμασταν ότι η εκτέλεση πραγματοποιήθηκε απροβλημάτιστα, είχαμε την πρώτη θετική ένδειξη για το αν είναι ουσιώδης η εκτέλεση του τροποποιημένου συστήματος αρχείων σε πραγματικό σύστημα.

Κατά τις δοκιμές στην εικονική μηχανή, αλλά και μια φορά στο μηχάνημα αξιολόγησης, παρατηρήθηκε έλλειψη αποκρισιμότητας μετά από πολύ σύντομο διάστημα χρήσης του τροποποιημένου συστήματος αρχείων. Το journal δεν υποδείκνυε τον λόγο του προβλήματος, και δεν εμφανιζόταν κάποιο χρήσιμο διαγνωστικό στην κονσόλα του QEMU. Το πρόβλημα αποτελεί προφανή περίπτωση έλλειψης ευστάθειας. Ωστόσο, ήταν αρκετά σπάνια και απρόβλεπτη

η παρουσία του, που έκανε την αποσφαλμάτωση του απρόσιτη στα χρονικά πλαίσια της εργασίας. Πραγματοποιήθηκε προσπάθεια να διαμορφωθούν συνθήκες πρόκλησης του προβλήματος μέσω πολύωρης χρήσης του συστήματος αρχείων και έγκαιρης διάγνωσης του σφάλματος μέσω του GDB, χωρίς καμία επιτυχία.

7.2 Αξιολόγηση Ορθότητας

Η ιδιότητα πάνω στην οποία δομήσαμε τις αλλαγές μας υπενθυμίζεται πως είναι η κατά NUMA τοπικότητα των δεσμεύσεων χώρου. Είναι προφανές ότι χρειαζόμαστε μια σειρά ελέγχων που θα εξετάζουν την τήρηση της ιδιότητας αυτής, ιδίως για την αποσφαλμάτωση κατά την ανάπτυξη. Υπήρξαν πολλές φορές που δεν ήταν εμφανής η επίδραση κάποιας προϋπάρχουσας βελτιστοποίησης στην κατά NUMA τοπικότητα μέχρι που προστέθηκε ο κατάλληλος έλεγχος ορθότητας και οδήγησε στον εντοπισμό της. Η συνάρτηση `numa_test_awareness` του αρχείου `experiments/scripts/numa/numa_utilities.sh` του πηγαίου κώδικα της εργασίας αναδεικνύει ακριβώς πως υλοποιήθηκε ο έλεγχος ορθότητας στο σύνολο του.

7.2.1 Έλεγχος ορθότητας μέσω του FIO

Ο πρώτος και πιο απλοϊκός τρόπος να ελέγξουμε την ορθότητα των αλλαγών είναι μέσω ενός κατάλληλα προσαρμοσμένου πειράματος με το FIO. Μια προφανής διαμόρφωση τέτοιου πειράματος αποτελεί η ισοκατανομή ενός αριθμού νημάτων στους επιμέρους NUMA κόμβους, με το καθένα να δημιουργεί και να κάνει προσβάσεις σε ένα δικό του αρχείο. Αυτό γίνεται εύκολα με δύο επιμέρους εργασίες, μια για κάθε κόμβο. Η παράμετρος που θέλει ιδιαίτερη προσοχή είναι η `create_serialize`. Αν αφεθεί στην προεπιλεγμένη τιμή της, τότε η δημιουργία των αρχείων θα γίνει από το βασικό νήμα του FIO. Επομένως, η τοποθέτηση των νημάτων θα αποβεί ασυσχέτιστη με αυτήν των αρχείων, και το πείραμα δεν θα έχει νόημα. Είναι επίσης σημαντικό να έχουν διαγραφεί προϋπάρχοντα αρχεία του πειράματος.

Μετά την εκτέλεση της εργασίας του FIO μένει να ελέγξουμε ότι τα αρχεία τοποθετήθηκαν στον σωστό κόμβο. Είναι εύκολο να ξεχωρίσουμε για ποιον κόμβο προοριζόταν κάθε αρχείο, καθώς αυτά ονοματίζονται βάσει της ονόματος της αντίστοιχης εργασίας του FIO. Θέλουμε πλέον έναν τρόπο εύρεσης το κόμβου του αρχείου στο userspace.

Εκμεταλλευόμαστε την ευκολία που μας παρέχει το γεγονός ότι έχουμε μόνο 2 NUMA κόμβους στο σύστημα αξιολόγησης. Επιτυγχάνουμε κατά προσέγγιση το ζητούμενο βρίσκοντας το πρώτο group του δεύτερου κόμβου μέσω του

`dumpe2fs`, και συγκρίνοντας το με το group στο οποίο ανήκει το πρώτο block του αρχείου, το οποίο μπορούμε να το λάβουμε μέσω του `debugfs`.

Είναι ευνόητο ότι δεν αρκεί να ελέγξουμε μόνο το πρώτο block του αρχείου, αλλά πολλές φορές η τοποθέτηση του αποτελεί ισχυρή ένδειξη για την τήρηση της κατά NUMA τοπικότητας, και άλλωστε επιθυμούμε ο έλεγχος να είναι όσο πιο γρήγορος γίνεται ώστε να μπορούμε να προχωράμε απρόσκοπτα την ανάπτυξη του πυρήνα. Ο σχετικός κώδικας σε bash φαίνεται στα αρχεία `experiments/scripts/numa/numa_test_fio.sh` και `experiments/scripts/numa/numa_common.sh`.

7.2.2 Έλεγχος ορθότητας μέσω του Filebench

Το FIO μας δίνει έναν άμεσο τρόπο ελέγχου της ορθότητας των αλλαγών μας. Δεν διαθέτει όμως τον απαραίτητο βαθμό πολυπλοκότητας στις προσβάσεις αλλά ούτε δυναμικότητα στην τοποθέτηση των νημάτων ώστε να βεβαιωθούμε ότι η ιδιότητα της κατά NUMA τοπικότητας στις δεσμεύσεις επιμένει και σε πιο σύνθετες εργασίες. Αυτό ακριβώς το πρόβλημα μπορεί να λύσει το εργαλείο Filebench. Όπως είχαμε αναφέρει στο κεφάλαιο *Μεθοδολογία*, το εργαλείο αυτό μπορεί να προσομοιώσει την συμπεριφορά διάφορων περιπτώσεων προγραμμάτων ως προς το πως εκτελούν I/O.

Το σημαντικότερο χαρακτηριστικό του Filebench σε αυτήν την περίπτωση αποτελεί το γεγονός ότι ορίζει ένα σύνολο αρχείων, εκ των οποίων μερικά έχουν προδεσμευτεί, και κάποια άλλα αν επιλεχτούν τυχαία κατά την εκτέλεση της εργασίας θα δεσμευτούν επιτόπου. Αυτή η τυχαιότητα στην δημιουργία αρχείων μπορεί να οδηγήσει στην διαμόρφωση ενός ελέγχου που θα μας πιστοποιεί με μεγαλύτερη βεβαιότητα την ευρωστία των αλλαγών που εξετάζουμε.

Το Filebench από μόνο του δεν περιέχει την έννοια της NUMA τοπολογίας στον σχεδιασμό του, εννοώντας ότι δεν υπάρχει κανένας τρόπος να οριστεί προτίμηση στην τοποθέτηση σε κόμβους των νημάτων που δημιουργούνται. Επίσης, υπενθυμίζεται ότι για να ελέγξουμε εκ των υστέρων αν έγιναν σωστά οι δεσμεύσεις των αρχείων, πρέπει να ορίσουμε μια αντιστοιχία μεταξύ του ονόματος κάθε αρχείου και του κόμβου στον οποίον θέλουμε να τοποθετηθεί.

Για τους λόγους που αναφέρθηκαν, τροποποιούμε ελάχιστα τον πηγαίο κώδικα του Filebench ώστε κατά την δημιουργία του αρχείου να περιορίζεται η εκτέλεση του νήματος στον κόμβο που υποδεικνύει το υπόλοιπο του αναγνωριστικού του αρχείου με τον συνολικό αριθμό των κόμβων. Για παράδειγμα, έχοντας δύο κόμβους, τα αρχεία των οποίων το όνομα αντιστοιχεί σε άρτιο νούμερο δεσμεύονται στον κόμβο 0, και τανάπαλιν. Οι απαραίτητες τροποποιήσεις φαίνονται στο αρχείο `patches/filebench_test.patch`, το οποίο και είναι απαραίτητο να εφαρμοστεί στον πηγαίο κώδικα του Filebench προτού πραγματοποιηθεί ο αντίστοιχος έλεγχος.

Η ρουτίνα ελέγχου ορθότητας μέσω του Filebench είναι διαθέσιμη στο αρχείο `experiments/scripts/numa/numa_test_filebench.sh`. Η εκτέλεση της αναδεικνύει ότι τα ποσοστά λάθους στην τοποθέτηση αρχείων μειώνεται από 50% για την αρχική εκδοχή του ext4, σε λιγότερο από 5% για την τροποποιημένη εκδοχή του. Σίγουρα το γεγονός ότι το ποσοστό δεν καταλήγει μηδενικό είναι αποθαρρυντικό, αλλά θεωρείται στα όρια του επιτρεπτού δεδομένου ότι προσπαθούμε να αναδείξουμε ποιοτικά την επίδραση της κατά NUMA τοπικότητας στην επίδοση, και δικαιολογείται από την αυξημένη πολυπλοκότητα του κώδικα του ext4.

7.2.3 Έλεγχος ορθότητας αποκλειστικά μέσω bash script

Ο έλεγχος αυτός μοιάζει αρκετά με αυτόν του FIO, με την διαφορά ότι η δημιουργία των αρχείων γίνεται απευθείας στο bash script μέσω του τελεστή ανακατεύθυνσης και της εντολής `taskset` για την επιλογή του αντίστοιχου κόμβου. Κοινώς, η δημιουργία των αρχείων μοιάζει ως εξής:

```
$ taskset -a -c <cpu_id> bash -c "./scripts/bigfile.sh >
    ↪ /mnt/pmem/file_<cpu_id>_\$i"
```

Θα περιμέναμε ότι αν πετυχαίνει ο έλεγχος μέσω του FIO, θα πετυχαίνει και ο έλεγχος με την δημιουργία μέσω ανακατεύθυνσης. Παραδόξως όμως αυτό φαίνεται να μην ισχύει, τουλάχιστον όχι πάντα. Η διερεύνηση του θέματος αυτού πραγματοποιήθηκε στο παράρτημα *Διερεύνηση της συμπεριφοράς του τελεστή ανακατεύθυνσης του Bash shell.*

Έχει ενδιαφέρον αυτή η περίπτωση, κυρίως γιατί παρουσιάζει την ανάγκη διερεύνησης για το πότε η μη διαθεσιμότητα κατάλληλων kernel worker μπορεί να βλάψει την κατά NUMA τοπικότητα. Έλεγχοι που έχουν γίνει μέσω του FIO σε κατάλογο εγγραφής που έχει απενεργοποιημένο το χαρακτηριστικό DAX, έχουν δείξει ότι υπό κανονικές συνθήκες εγγραφής, δηλαδή μέσω syscall προς το VFS, υπάρχει από ένας kernel worker για κάθε NUMA κόμβο. Κοινώς. παρακάμπτεται η πηγή του προβλήματος που διακρίναμε με την προαναφερθείσα διερεύνηση και δεν πλήττεται η ορθότητα του σχεδιασμού μας σε αυτήν την περίπτωση.

Άλλο ενδιαφέρον γεγονός είναι ότι σε αρχικά στάδια των τροποποιήσεων του ext4, αυτή η ιδιοτροπία του υπό εξέταση ελέγχου δεν ήταν εμφανής. Αυτό συνέβαινε καθώς η επιλογή του κόμβου εγγραφής γινόταν βάσει του inode του αρχείου, για λόγους ευκολίας στην αποσφαλμάτωση. Δηλαδή, ο κόμβος στον οποίον γινόταν η δέσμευση μεταδεδομένων του αρχείου ήταν αυτός στον οποίον πραγματοποιούνταν όλες οι εγγραφές στο εξής, αντί για τον κόμβο του νήματος που έκανε την αντίστοιχη επέκταση του αρχείου. Εφόσον λοιπόν η δέσμευση

μεταδεδομένων γινόταν σωστά, οι εγγραφές γίνονταν στον ίδιο με αυτά κόμβο, και ο έλεγχος φαινόταν να είναι επιτυχής. Αναδεικνύεται λοιπόν η σημασία που έχει η προσεκτική διαμόρφωση και ανάπτυξη της μεθοδολογίας αξιολόγησης.

7.3 Έλεγχος Αποδοτικότητας

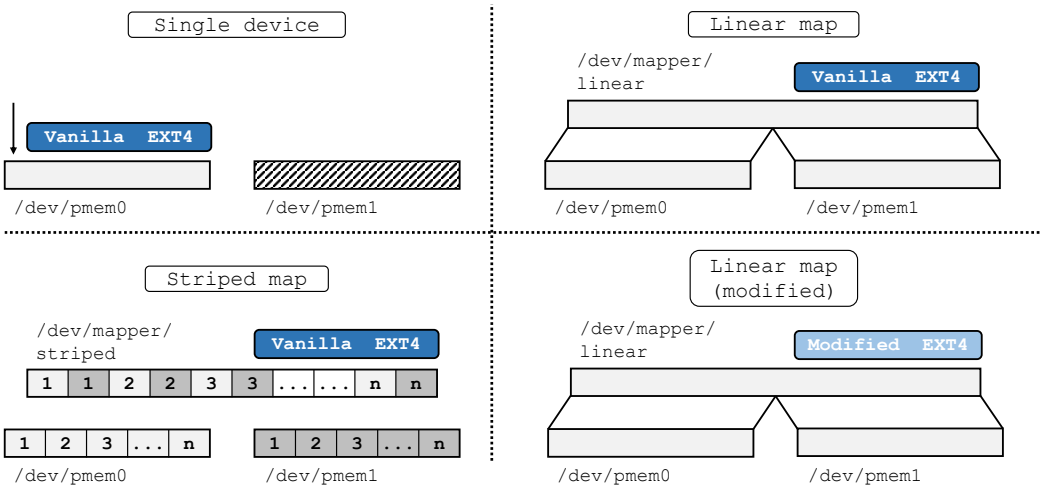
Έχοντας επαληθεύσει την ορθότητα και την σχετική ευστάθεια των αλλαγών που πραγματοποιήθηκαν στο ext4, ήρθε η στιγμή να εξετάσουμε σε ποιον βαθμό ανορθώθηκαν τα θέματα επίδοσης που παρουσιάστηκαν στην ενότητα *Πειράματα*. Θα εξετάσουμε αρχικά τι επιτάχυνση λαμβάνουμε για απλά microbenchmark μέσω του FIO, και στην συνέχεια θα πραγματοποιήσουμε μια αξιολόγηση για πιο σύνθετες εργασίες μέσω του Filebenh. Σε όλες τις μετρήσεις, η μετρική που εξετάζουμε είναι το bandwidth.

Είναι πολύ σημαντικό να αναφερθεί ότι η εναλλαγή μεταξύ αρχικού και τροποποιημένου ext4 για τους σκοπούς των μετρήσεων γίνεται μέσω της παραμέτρου προσάρτησης `numa` που προσθέσαμε. Πραγματοποιήθηκε εκτεταμένη προσπάθεια ώστε οι αλλαγές να έχουν μηδενική επίδραση όταν δεν δίνεται η παράμετρος `numa`, δηλαδή να έχουμε ισοδύναμη συμπεριφορά με την αρχική εκδοχή του ext4. Ωστόσο, το σωστό θα ήταν να έχουμε ένα ξεχωριστό σύστημα αρχείων με τις αλλαγές, ακριβώς όπως έγινε για το σύστημα αρχείων NOVA για την δημοσίευση του WineFS [21]. Λόγω όμως έλλειψης χρόνου, και βάσει κάποιων άτυπων ελέγχων που δείχνανε ότι η λειτουργία της παραμέτρου `numa` ήταν η αναμενόμενη, προτιμήθηκε η εναλλακτική.

Αναφέρουμε σε πρώτο στάδιο τις πιθανές διαφορετικές διαμορφώσεις συσκευών και του συστήματος αρχείων, τις οποίες θα αξιοποιήσουμε για τον πειραματισμό μας. Επιγραμματικά, έχουμε:

- **Single device**: αξιοποιούμε μόνο μια συσκευή, π.χ. την `/dev/pmem0` και την αρχική εκδοχή του ext4.
- **Linear map**: Συνενώνουμε γραμμικά τις συσκευές `/dev/pmem0` και `/dev/pmem1` μέσω του device mapper, και τοποθετούμε στην γραμμική συσκευή την αρχική εκδοχή του ext4.
- **Striped map**: Συνενώνουμε κατά τμήματα τις συσκευές `/dev/pmem0` και `/dev/pmem1` μέσω του device mapper με μέγεθος τμήματος (chunk size) ίσο με το μέγεθος σελίδας, δηλαδή 4KB. Πάνω από την τμηματική συνένωση έχει νόημα να τοποθετήσουμε μόνο την αρχική εκδοχή του ext4, το οποίο και κάνουμε. Υπενθυμίζεται ότι οι τροποποιήσεις που έγιναν στο ext4 βασίζονται στην υπόθεση της γραμμικής συνένωσης συσκευών.

- **Linear map (modified)**: Έχουμε γραμμική συνένωση των συσκευών `/dev/pmem0` και `/dev/pmem1`, και τοποθετούμε πάνω από την γραμμική συσκευή την τροποποιημένη εκδοχή του ext4. Αυτή η διάταξη είναι η μόνη αντιπροσωπευτική της δουλείας που έγινε σε αυτήν την εργασία.



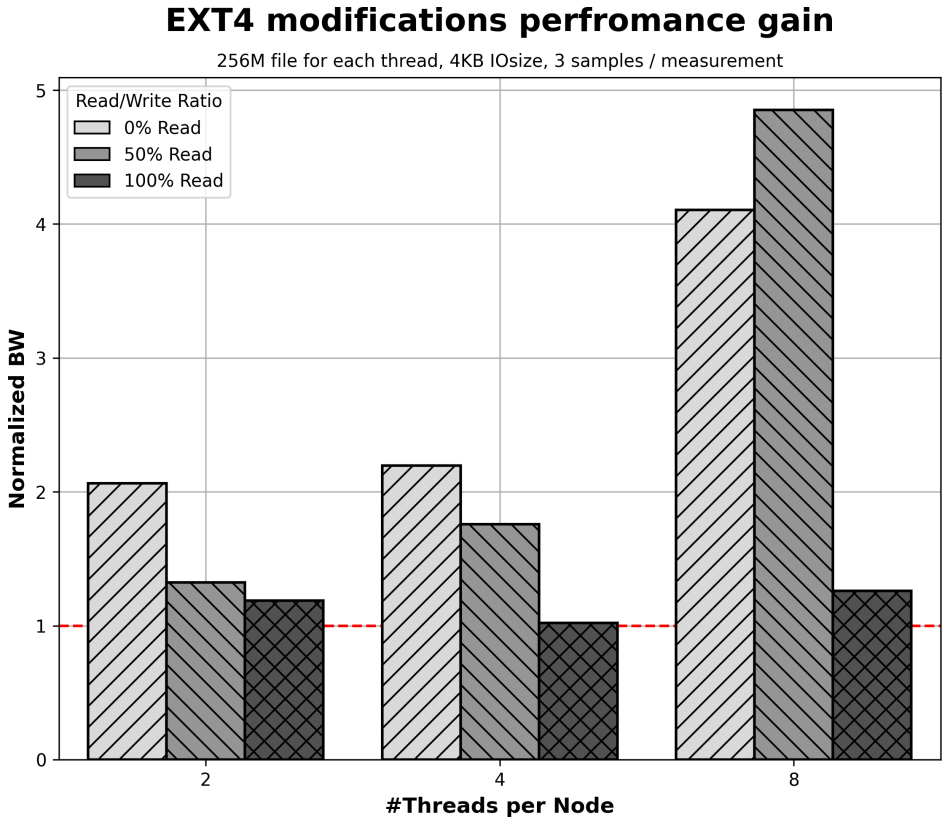
Σχήμα 7.1: Οι διαφορετικές διατάξεις που χρησιμοποιούνται για την αξιολόγηση

7.3.1 Έλεγχος αποδοτικότητας μέσω του FIO

Κατασκευάζουμε ένα απλό πείραμα μέσω του FIO, στο οποίο κατανέμουμε ομοιόμορφα τα νήματα μεταξύ των NUMA κόμβων, και το καθένα εργάζεται πάνω σε δικό του αρχείο 256MB πραγματοποιώντας προσβάσεις με 0%, 50% ή 100% ποσοστό αναγνώσεων. Οι προσβάσεις γίνονται με τρόπο λειτουργίας DAX. Οι παράμετροι εκτέλεσης του FIO στις οποίες εστιάζουμε αποτελούν ο αριθμός των νημάτων ανά κόμβο και το ποσοστό των προσβάσεων που είναι αναγνώσεις, η οποία είναι ενιαία για τα νήματα. Παρουσιάζουμε τα αποτελέσματα εκτέλεσης του πειράματος σε διάταξη Linear map (modified), κανονικοποιημένη προς τα αποτελέσματα της διάταξης Linear map. Έτσι, αποσκοπούμε να αναδειχθεί το κέρδος στην επίδοση που επιφέρεται από την δουλεία της ενότητας *Υλοποίηση*. Το πείραμα περιγράφεται στο αρχείο `experiments/scenarios/test_mods.sh`.

Σημειώνεται ότι είτε χρησιμοποιήσουμε ως βάση κανονικοποίησης την διάταξη Linear map, είτε την διάταξη Single device, λαμβάνουμε ίδια αποτελέσματα ποιοτικά, εφόσον το συνολικό μέγεθος των αρχείων του πειράματος καλύπτεται με άνεση από τον χώρο μιας μόνο συσκευής. Επίσης, για να έχει νόημα

το πείραμα, προσέχουμε ώστε η παράμετρος `create_serialize` του FIO να έχει τεθεί όπως πρέπει στην τιμή 0.

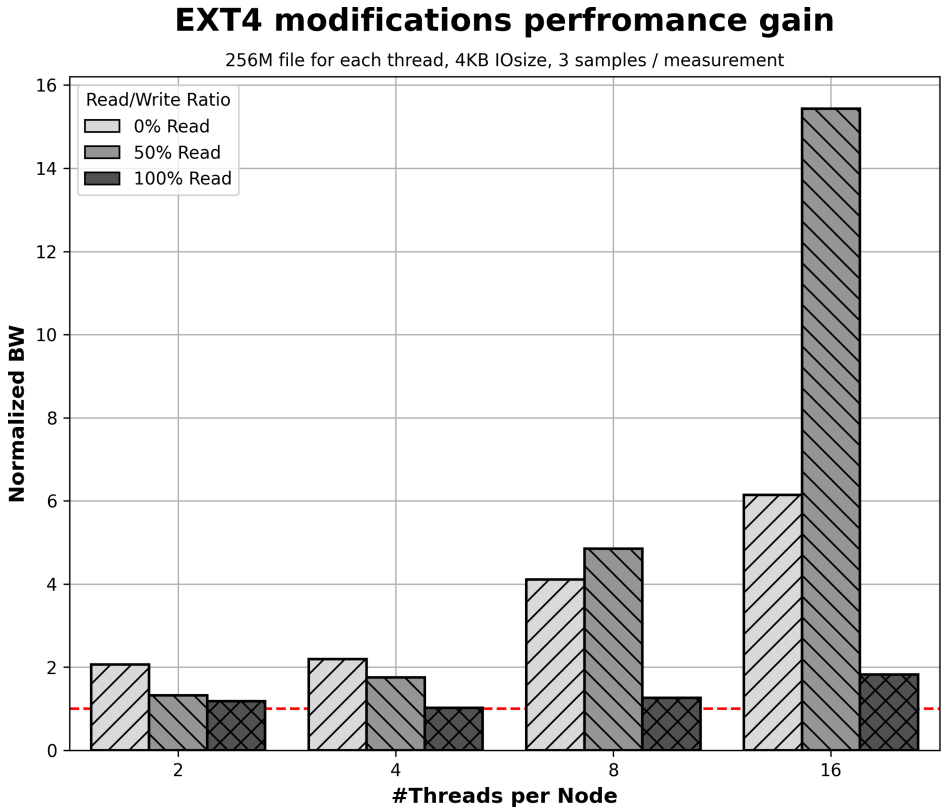


Σχήμα 7.2: Αποτελέσματα αξιολόγησης μέσω του FIO (μέχρι 8 νήματα ανά NUMA κόμβο)

Στο σχήμα 7.2 παρουσιάζονται τα αποτελέσματα που λαμβάνουμε από το πείραμα. Πρώτο συμπέρασμα που εξάγουμε, είναι ότι αναμενόμενα οι καθαρές εγγραφές επωφελούνται άμεσα από την ιδιότητα της NUMA τοπικότητας. Από την άλλη, οι καθαρές αναγνώσεις έχουν αναμενόμενα αμελητέο κέρδος στην επίδοση, αφού οι αλλαγές που κάναμε δεν αφορούν άμεσα αυτό το είδος πρόσβασης.

Αυτό που προκαλεί ιδιαίτερη εντύπωση είναι η επιτάχυνση που λαμβάνουμε για τις μικτές προσβάσεις, η οποία τείνει να είναι μεγαλύτερη και από την επιτάχυνση που λαμβάνουμε για τις καθαρές εγγραφές. Αυτό οφείλεται στην καταπολέμηση του φαινομένου "Read After Remote Write" που καταφέρνουμε μέσω των τοπικών δεσμεύσεων. Μάλιστα, όπως φαίνεται στο σχήμα 7.3, στο οποίο έχουμε συμπεριλάβει μετρήσεις για 16 νήματα ανά κόμβο, η βελτίωση φαίνεται να είναι αξιοσημείωτη, και ο σχεδιασμός μας τελικά καταφέρνει να

αντιμετωπίσει τις καταγεγραμμένες παθογένειες των μη πτητικών μνημών σε NUMA συστήματα.



Σχήμα 7.3: Αποτελέσματα αξιολόγησης μέσω του FIO (μέχρι 16 νήματα ανά NUMA κόμβο)

7.3.2 Έλεγχος Αποδοτικότητας μέσω του Filebench

Σε επόμενο στάδιο θέλουμε να εξετάσουμε την συμπεριφορά του τροποποιημένου συστήματος αρχείων σε συνθήκες πιο περίπλοκων workload. Προσφέρεται για αυτόν τον σκοπό το μετροπρόγραμμα Filebench.

Όπως έχει αναφερθεί και προηγουμένως, το Filebench δεν περιέχει την έννοια της αρχιτεκτονικής NUMA στον σχεδιασμό του. Αυτό είναι πολύ λογικό, αφού σκοπός του είναι να αποτελεί εργαλείο εξέτασης της επίδοσης των βασικών διεπαφών που προσφέρει ένα σύστημα αρχείων. Το τροποποιημένο σύστημα αρχείων της παρούσας εργασίας λαμβάνει ως υπόδειξη δέσμευσης δεδομένων τον κόμβο από τον οποίον εκτελείται το αντίστοιχο νήμα, κάτι το οποίο αποτελεί ιδιάζουσα περίπτωση. Επομένως, το Filebench δεν έχει παραμέτρους

για την επιλογή κόμβου προδέσμευσης των αρχείων, ούτε για την τοποθέτηση των νημάτων εκτέλεσης στους κόμβους.

Προσθήκες στο Filebench

Προκειμένου να επεκτείνουμε τις δυνατότητες του εργαλείου για τους σκοπούς της εργασίας, έγινε μια σειρά από προσθήκες που καταγράφονται στο αρχείο `experiments/patches/filebench_numa.patch`. Στην συνέχεια παρουσιάζεται η συλλογιστική πίσω από τις αλλαγές, και τι προσθέτουν στην λειτουργία του filebench.

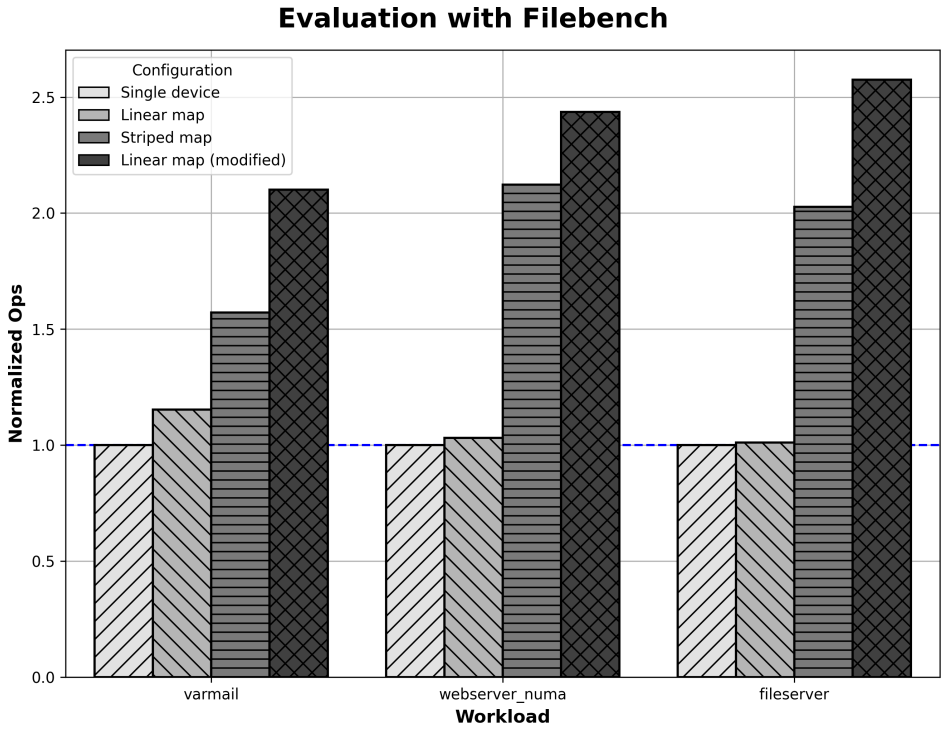
Η προδέσμευση των αρχείων σκοπεύει να δημιουργήσει μια κατάσταση στην οποία θα μπορούσε να έχει επέλθει το πρόγραμμα από ένα προηγηθέν διάστημα εκτέλεσης. Θεωρούμε ότι ένα νήμα έχει την τάση να εκτελείται περισσότερο σε έναν συγκεκριμένο κόμβο, κάτι το οποίο είναι πολύ πιθανό, είτε λόγω χρήσης taskset για καλύτερη αξιοποίηση της L3 Cache σε κάθε κόμβο, είτε δεδομένου της χρήσης ενός συστήματος όπως το userspace component που περιγράψαμε. Επομένως, στο υποθετικό προηγηθέν διάστημα εκτέλεσης, το σύνολο αρχείων ενός νήματος θα έχει την τάση να είναι συγκεντρωμένο σε έναν κόμβο.

Βάσει των όσων αναφέρθηκαν, καταλαβαίνουμε ότι έχει νόημα να προσθέσουμε μια προαιρετική παράμετρο numa στα fileset του Filebench, η οποία λαμβάνει μια ακέραια τιμή που αντιστοιχεί σε αναγνωριστικό κόμβου. Σε συνθήκες απουσίας της έχουμε ίδια συμπεριφορά με αυτήν της αρχικής εκδοχής του filebench. Δοθέντος αυτής, τοποθετούμε το νήμα προδέσμευσης στον κόμβο που έχει προσδιοριστεί, προκειμένου να συγκεντρωθούν εκεί τα αρχεία του fileset.

Επίσης, προσθέτουμε μια αντίστοιχη προαιρετική παράμετρο numa για το πεδίο thread ενός process, που προσδιορίζει τον NUMA κόμβο στον οποίον τοποθετούνται τα νήματα που ορίζονται από το πεδίο. Και αυτή η παράμετρος έχει νόημα, καθώς θέλουμε να εξετάσουμε την επίδραση στην επίδοση που μπορεί να επιφέρει η στοχευμένη τοποθέτηση νημάτων, όπως αυτή θα μπορούσε να επιτευχθεί με μια υποδομή όπως το userspace component.

Ρύθμιση και ορισμός των workload

Οι προσαρμογές που πραγματοποιήθηκαν στα workload για την εκτέλεση στο μηχάνημα αξιολόγησης περιέχονται στο αρχείο `experiments/patches/filebench_workloads.patch`. Η βασική διαφοροποίηση στα προϋπάρχοντα workload είναι ο προσδιορισμός ως καταλόγου εργασίας ενός DAX καταλόγου στο σημείο προσάρτησης του τροποποιημένου συστήματος αρχείων (συνήθως ο `/mnt/pmem/daxdir`), και η αύξηση του συνολικού αριθμού αρχείων. Στο varmail συγκεκριμένα αυξήσαμε το μέσο μέγεθος επέκτασης ενός αρχείου



Σχήμα 7.4: Αποτελέσματα αξιολόγησης μέσω του Filebench

σε 256KB, ώστε να αναδειχθεί πιο έντονα η επίδραση των NUMA aware δεσμεύσεων. Αυτό δεν είναι αβάσιμο, καθώς θα μπορούσε μέσω του διακομιστή να μεταφέρονται και συνημμένα αρχεία εκτός από απλό κείμενο.

Έχουν προστεθεί και δύο καινούργια workload, τα `fileserver_numa` και `webserver_numa`, τα οποία λειτουργούν ακριβώς όπως τα `fileserver` και `webserver` με την διαφορά ότι εκτελούνται παράλληλα δύο instance για το καθένα, με κάθε instance να έχει δικό του fileset και να εκτελείται σε διαφορετικό NUMA κόμβο. Θα χρησιμοποιήσουμε το `weberserver_numa` workload αντί για το ίδιο το webserver καθώς είναι επικεντρωμένο στις αναγνώσεις, στις οποίες επωφελείται το τροποποιημένο ext4 μόνο όταν έχει προηγηθεί NUMA aware δέσμευση του προς ανάγνωση αρχείου. Διαφορετικά, δεν θα βλέπαμε σημαντικές διακυμάνσεις μεταξύ των διαφορετικών διατάξεων εκτέλεσης.

Στο σχήμα 7.4 παρουσιάζονται τα αποτελέσματα από την εκτέλεση του πειράματος του αρχείου `experiments/scenarios/filebench.sh`. Επίσης, στον πίνακα 7.1 παρατίθενται οι ποσοστιαίες αυξήσεις στην επίδοση της διάταξης Linear map (modified) σε σχέση με τις Striped map και Linear map.

workload	Βελτίωση έναντι του Linear map (%)	Βελτίωση έναντι του Striped map (%)
varmail	82	34
webserver_numa	136	14
fileserver	154	27

Πίνακας 7.1: Βελτίωση επίδοσης στα επιλεγμένα workload του Filebench

Ανάλυση των αποτελεσμάτων

Βλέπουμε ότι η διάταξη που αντιπροσωπεύει την τροποποιημένη εκδοχή του ext4 παρουσιάζει την καλύτερη επίδοση σε κάθε περίπτωση. Οι επιδόσεις των διατάξεων Single device και Linear map διαφέρουν ελάχιστα μεταξύ τους, διότι λόγω του μεγέθους και του αριθμού των αρχείων στην Linear map καταλήγει να αξιοποιείται πολύ περισσότερο η μια συσκευή σε σχέση με την άλλη.

Το Striped map έχει συγκρίσιμες επιδόσεις με το Linear map (modified). Ο λόγος που το έχουμε συμπεριλάβει ως διάταξη, είναι ότι το Striped map είναι κάτι που προκύπτει άμεσα χωρίς την δουλειά της παρούσας εργασίας. Επίσης, έχει προοπτική να δουλέψει καλύτερα από τις διατάξεις Single device και Linear map, γιατί μπορεί και εκθέτει σε επίπεδο μεμονωμένων προσβάσεων το αθροιστικό bandwidth όλων των συσκευών, ασχέτως εάν όπως θα αναφέρουμε αυτό υποχρησιμοποιείται. Επομένως, οφείλουμε να συγκρίνουμε την δουλεία της εργασίας αυτής με όποια προϋπάρχουσα, εύκολη και σχετικά αποδοτική λύση.

Μέσω του `varmail` αναδεικνύεται το συγκριτικό πλεονέκτημα του τροποποιημένου ext4 για μεγάλες επεκτάσεις σε αρχεία, οι οποίες επεκτάσεις αναπαριστούν απαντήσεις σε μηνύματα. Από αυτό το workload λαμβάνουμε και την μεγαλύτερη διαφοροποίηση σε σχέση με την διάταξη Striped map, ακριβώς λόγω του προαναφερθέντος πλεονεκτήματος. Ωστόσο, δεν φαίνεται να υπάρχει κάτι στην μορφή της αλληλουχίας ενεργειών I/O που να προσφέρει διαφοροποίηση μεταξύ των διατάξεων όταν αναιρούμε τις επεκτάσεις αρχείων. Επομένως θεωρούμε ότι ο αποφασιστικός παράγοντας στο κέρδος επίδοσης αποτελούν σχεδόν αποκλειστικά οι επεκτάσεις.

Στο `weberver_numa` επωφελούμαστε σε μεγάλο βαθμό από τις τροποποιήσεις. Το ενδιαφέρον είναι ότι η διαφοροποίηση με την διάταξη Striped map είναι μικρή, αλλά όχι αμελητέα. Στην προαναφερθείσα λαμβάνουμε πλεονέκτημα επίδοσης γιατί έχουμε αρχεία στα οποία γίνονται αποκλειστικά αναγνώσεις, και η ταυτόχρονη πρόσβαση σε πολλαπλούς NUMA κόμβους εν προκειμένω περισσότερο εκθέτει το αθροιστικό bandwidth των συσκευών παρά επενεργεί ζημιογόνα. Αυτό αναδεικνύεται λογικό από επισκόπηση του σχήματος 4.1, στο

οποίο φαίνεται ότι αν έχουμε μόνο αναγνώσεις, το bandwidth τοπικής πρόσβασης διαφοροποιείται λίγο σε σχέση με αυτό της απομακρυσμένης. Εκεί που διαφοροποιείται περισσότερο η τροποποιημένη διάταξη, είναι η επέκταση του αρχείου καταγραφής (log file).

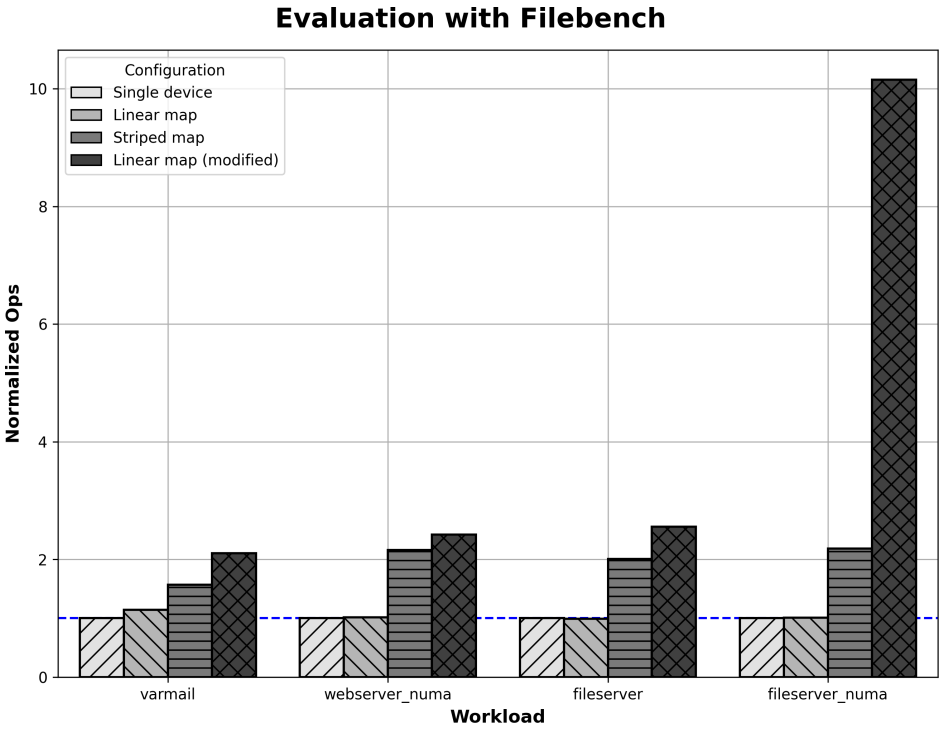
Τέλος, στο `fileserver` παρατηρείται το μεγαλύτερο κέρδος επίδοσης σε σχέση με τις διατάξεις Signle device και Linear map. Επίσης, διαφοροποιείται έντονα και από το Striped map, και ένας μη αμελητέος παράγοντας αυτού φαίνεται να είναι η γρήγορη ανάκτηση μεταδεδομένων που επιζητεί η εντολή `stat`, το οποίο επιτυγχάνεται λόγω της κατά NUMA τοπικότητας των μεταδεδομένων στον σχεδιασμό μας.

Περαιτέρω ανάλυση

Αυτό που έχει το μεγαλύτερο ενδιαφέρον είναι η δυνητική επίδραση που έχει η κατανομή των block ενός αρχείου μεταξύ κόμβων λόγω επέκτασης είτε από διαφορετικά, είτε από μετακινούμενα νήματα. Στα έως τώρα αποτελέσματα δεν διαφαίνεται η επίδραση αυτή, ωστόσο θα είχε ενδιαφέρον η συγγραφή κάποιου microbenchmark που στοχεύει στην ανάλυση της.

Επιπροσθέτως, έχει σημασία η σχετική τοποθέτηση μεταδεδομένων και δεδομένων ενός αρχείου σε διαφορετικούς κόμβους, και πως μπορεί να επηρεάσει την επίδοση. Στο varmail μετά από έντονη παραμετροποίηση κατά την διάρκεια πειραματισμού, που δυστυχώς δεν καταγράφηκε επαρκώς, είχε παρατηρηθεί έντονη υποβάθμιση της επίδοσης σε οποιαδήποτε διάταξη περιλαμβάνει περισσότερους από έναν κόμβους. Ο κύριος λόγος υποπτεύθηκε ότι ήταν ο συγχρονισμός μεταδεδομένων μέσω της εντολής fsync, που επιφέρει πολλαπλά μικρά αιτήματα απομακρυσμένων εγγραφών, και επαναφέρει πιθανώς την παθογένεια "Read After Remote Write." Επομένως θα είχε μεγάλο ενδιαφέρον να εξεταστεί ένας μηχανισμός αποδοτικού συγχρονισμού μεταδεδομένων σε μελλοντική επέκταση της εργασίας.

Τέλος, αν και το Striped map φαίνεται να αποτελεί προσεγγιστικά το ίδιο καλή λύση με την τροποποιημένη διάταξη που προκύπτει από το έργο αυτής της εργασίας, στην πραγματικότητα η επίδοση της είναι υπό μια έννοια φραγμένη. Δηλαδή, αν και αξιοποιεί ταυτοχρόνως το bandwidth πολλαπλών συσκευών, αυτό γίνεται με τρόπο που επιβάλει πάντα ομοιόμορφη πρόσβαση στις συσκευές, και οι όποιες παθογένειες των απομακρυσμένων προσβάσεων δεν μπορούν να τεθούν υπό έλεγχο. Επομένως, μπορούμε να το αναλογιστούμε ως υποαξιοποίηση του αθροιστικού bandwidth. Το τροποποιημένο σύστημα αρχείων διατηρεί έναν σαφή διαχωρισμό μεταξύ των συσκευών, και με μια λογική τοποθέτηση των νημάτων εκτέλεσης μπορεί να οδηγήσει σε θεαματικό κέρδος, όπως φαίνεται στο διάγραμμα του σχήματος 7.5, στο οποίο έχει συμπεριληφθεί το workload fileserver_numa.



Σχήμα 7.5: Αποτελέσματα αξιολόγησης μέσω του Filebench (συμπεριλαμβανομένου του workload `fileserver_numa`)

Η επιτάχυνση είναι τόσο μεγάλη γιατί το `fileserver` workload ορίζει 50 νήματα, τα οποία πρέπει να ξεπερνούν κατά πολύ το σημείο κορεσμού του bandwidth των μνημών κάθε κόμβου. Έχοντας σαφή διαχωρισμό μεταξύ των δύο παράλληλων διεργασιών του workload `fileserver_numa`, φαίνεται ότι οδηγούμαστε σε ανόρθωση πολλαπλών παθογενειών επίδοσης. Αυτό το φαινόμενο όμως είναι τόσο έντονο περισσότερο λόγω της κατανομής των νημάτων κάθε διεργασίας σε ξεχωριστούς κόμβους, το οποίο αναδεικνύει και την ανάγκη διαμόρφωσης του userspace component.

Κεφάλαιο 8

Σχετική Έρευνα

Μεταξύ της ανακοίνωσης των μη πτητικών μνημών της Intel και της εμπορικής διαθεσιμότητας τους, οι ερευνητές βασίστηκαν σε όσες πληροφορίες είχαν διαθέσιμες για την συμπεριφορά των προϊόντων αυτών, πραγματοποιώντας έργο με την βοήθεια συστημάτων προσομοίωσης για τους σκοπούς αξιολόγησης [30, 29]. Ένα σημαντικό παράγωγο αυτής της περιόδου υπήρξε το σύστημα αρχείων NOVA [29], το οποίο προσάρμοσε τεχνικές log-structuring (σειριακή εγγραφή (μέτα)δεδομένων σε κυκλικούς buffer) με τρόπο που εκμεταλλεύεται καλύτερα σε σχέση με συμβατικά συστήματα αρχείων τις γρήγορες τυχαίες προσβάσεις των μη πτητικών μνημών.

Πολλαπλές δουλειές [28, 21] έχουν επεκτείνει πρόχειρα το σύστημα αρχείων NOVA ώστε να υποστηρίζει δεσμεύσεις πάνω από πολλαπλούς NUMA κόμβους.

Αργότερα εμφανίστηκε το WineFS [18], ένα σύστημα αρχείων που επέκτεινε το PMFS της Intel προκειμένου οι δεσμεύσεις να διατηρούν την ικανότητα του συστήματος να παρέχει μεγάλες σελίδες (hugepages) ακόμα και υπό συνθήκες προχωρημένης γήρανσης. Αυτό βοηθάει σημαντικά την επίδοση σε προσβάσεις memory-mapped αρχείων.

Όσον αφορά την χρήση μη πτητικών μνημών σε συστήματα NUMA, η βιβλιογραφία είναι περιορισμένη. Το WineFS υποστηρίζει δημιουργία συστήματος αρχείων πάνω σε μνήμες πολλαπλών NUMA κόμβων. Η λογική NUMA awareness του WineFS βασίζεται στην μονοσήμαντη αντιστοιχία διεργασιών και κόμβων, και την μετακίνηση της διεργασίας στον επιλεγμένο κόμβο κάθε φορά που θέλει να κάνει κάποια εγγραφή.

Ενδιαφέρον επίσης για την έρευνα NUMA awareness σε συστήματα αρχείων παρουσιάζει η περίπτωση του "μετασυστήματος" αρχείων NapFS [17]. Αυτό ορίζει ανά κόμβο ένα επιλεγμένο σύστημα αρχείων, και σχηματίζει επιπλέον υποδομή μέσω βοηθητικών νημάτων και καθολικών δομών δεδομένων ώστε να προσφέρει δεσμεύσεις σε πολλαπλούς κόμβους με interleaved τρόπο.

Επιπλέον, έχουν πραγματοποιηθεί δουλειές που επιδιώκουν να αναδείξουν τα βασικά χαρακτηριστικά επίδοσης των μη πτητικών μνημών. Μια από αυτές [30] έχει αξιοποιηθεί εκτενώς στην βιβλιογραφία αλλά και ως πολύτιμη πηγή κατανόησης των ποιοτικών χαρακτηριστικών των μνημών για τους σκοπούς της παρούσας εργασίας. Άλλες δουλειές [16] πραγματοποιούν με παρόμοιο τρόπο ανάλυση των θεμελιωδών χαρακτηριστικών επίδοσης των συσκευών, τόσο ως μέσο αποθήκευσης όσο και ως μνήμη.

Κεφάλαιο 9

Επίλογος

9.1 Συμπεράσματα

Σκοπός μας με την εργασία αυτή ήταν να εξετάσουμε τις ιδιαιτερότητες της τρέχουσας τεχνολογίας μη πτητικών μνημών σε συστήματα μη ομοιόμορφης πρόσβασης μνήμης και πως μπορούμε να αναπροσαρμόσουμε κατάλληλα τις συναρτήσεις δεσμεύσεων (μέτα)δεδομένων του ext4 ώστε να σέβονται καλύτερα τις ιδιαιτερότητες αυτές. Πράγματι, είδαμε από την διαδικασία της πειραματικής αξιολόγησης ότι μια τέτοια προσέγγιση μπορεί τελικά να οδηγήσει σε σημαντικό κέρδος στον βαθμό εκμετάλλευσης της ρυθαμπόδοσης των μη πτητικών συσκευών.

Γενικότερα, αναδείξαμε εκ νέου ότι η στρατηγική ομοιόμορφης αξιοποίησης των πόρων ενός NUMA συστήματος μπορεί να μην αξιοποιεί το ίδιο αποδοτικά την συνολική επίδοση των διαθέσιμων πόρων σε σχέση με μια στρατηγική στην οποία έχουμε σαφή διαχωρισμό μεταξύ NUMA κόμβων και αξιοποιούμε κατά συνθήκες τον κατάλληλο κόμβο. Εν προκειμένω, η πρώτη στρατηγική αντιστοιχεί στην ενοποίηση όλων των μη πτητικών συσκευών σε μια striped λογική συσκευή, και η δεύτερη στρατηγική αναλογεί στον σχεδιασμό που παρουσιάσαμε στην ενότητα *Υλοποίηση*.

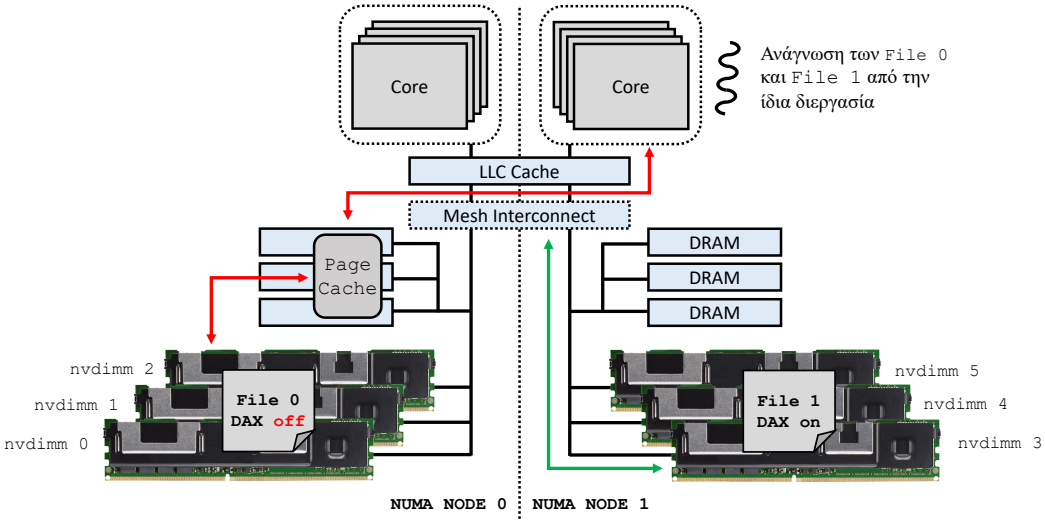
Όπως αναφέρθηκε σε διάφορα σημεία της εργασίας, η δουλειά αυτή επιτρέπει μια σειρά από ενδιαφέρουσες επεκτάσεις, τις οποίες θα αναφέρουμε (μερικές ανακεφαλαιωτικά) στην αμέσως επόμενη ενότητα.

9.2 Μελλοντικές Επεκτάσεις

Η αμεσότερη επέκταση της εργασίας αποτελεί η εξέλιξη του userspace component με τους τρόπους που περιγράφτηκαν στην υποενότητα *Βήματα επέκτασης του υπάρχοντος σχεδιασμού* της σχετικής ενότητας του κεφαλαίου

Υλοποίηση. Στο σύστημα αρχείων ext4 από την άλλη μπορούμε να προσθέσουμε αρχικά την υποστήριξη του χαρακτηριστικού `flex_bg` στον κώδικα του NUMA awareness, και ύστερα να εξετάσουμε την επίδραση των τροποποιήσεων στον τρόπο γήρανσης του συστήματος αρχείων. Συγκεκριμένα, χρειάζεται να εξεταστεί με περισσότερη λεπτομέρεια το κόστος στην επίδοση όταν έχουμε άναρχη κατανομή των block ενός αρχείου μεταξύ NUMA κόμβων όταν αυτό επιδέχεται επεκτάσεις από πολλαπλά νήματα σε διαφορετικούς κόμβους.

Εν συνεχεία παρουσιάζουμε πιο αναλυτικά δύο από τις σημαντικότερες και πιο περίπλοκες πιθανές επεκτάσεις.



Σχήμα 9.1: Τρόπος χρήσης του per-file DAX χαρακτηριστικού για εξομάλυνση των απομακρυσμένων προσβάσεων

9.2.1 Εξομάλυνση των απομακρυσμένων προσβάσεων μέσω του χαρακτηριστικού per-file DAX

Όπως έχει αναφερθεί προηγουμένως, ένας από τους βασικούς λόγους που επιλέχτηκε το σύστημα αρχείων ext4 προς επέκταση είναι η υποστήριξη ρύθμισης της λειτουργίας DAX, και μάλιστα ανά αρχείο. Δεδομένου αυτού, μπορούμε να δημιουργήσουμε ένα σύστημα στο οποίο χρησιμοποιούμε επιλεκτικά για κάποια δεδομένα την page cache και για κάποια άλλα την λειτουργία DAX ώστε να αξιοποιούμε το αθροιστικό bandwidth τόσο των μη πτητικών μνημών όσο και της RAM. Ένα τέτοιο σύστημα θα ανέθετε στην page cache τα αρχεία με μεγάλο βαθμό επαναληψιμότητας στις προσβάσεις, εφόσον έτσι εκμεταλλευόμαστε το μεγαλύτερο bandwidth των πτητικών συσκευών και αξίζει το κόστος αντιγραφής σελίδων στην page cache.

Το σύστημα που αναφέρθηκε θα μπορούσε να προσαρμοστεί ώστε να ενθαρρύνει την πρόσβαση σε αρχεία μέσω της page cache όταν αυτά επιδέχονται συχνές απομακρυσμένες προσβάσεις, προκειμένου να αποφευχθούν οι παθογένειες που παρατηρούνται στις απομακρυσμένες προσβάσεις μη πτητικών μνημών. Για να έχει νόημα ωστόσο η ενασχόληση με αυτήν την επέκταση, πρέπει να εξεταστεί πρώτα προσεκτικά πως αλληλεπιδρούν μεταξύ τους λειτουργίες άμεσης πρόσβασης και λειτουργίες μέσω της page cache, με εστίαση στο πως το write back στις μη πτητικές συσκευές μπορεί να έχει παρόμοια αρνητική επίδραση με την παθογένεια "Read After Remote Write."

9.2.2 Υποστήριξη Τοπικών Overwrite

Καταφέραμε σε αυτήν την εργασία να κατευθύνουμε τις δεσμεύσεις καινούργιων block ώστε να σέβονται την τοπολογία NUMA. Ωστόσο, με προφανή τρόπο οι εγγραφές πάνω σε ήδη δεσμευμένα block μπορούν να γίνουν μόνο στον NUMA κόμβο στον οποίον αντιστοιχεί το block. Αν τα δεδομένα προς εγγραφή έχουν επαρκώς μεγάλο μέγεθος, Θα μπορούσαμε να γράψουμε τις τροποποιημένες σελίδες προσωρινά σε κάποια block του τοπικού κόμβου και ύστερα να προσαρμόσουμε το extent tree ώστε να δείχνει στις τροποποιημένες σελίδες αντί για τις αρχικές στον απομακρυσμένο κόμβο.

Η υποστήριξη ενός τέτοιου χαρακτηριστικού έχει πολλές παραμέτρους προς εξέταση. Πρώτο παράδειγμα αποτελεί η ανάγκη αποδοτικής μεταφοράς των προς τροποποίηση σελίδων στον τοπικό κόμβο όταν αυτό είναι απαραίτητο. Ύστερα, χρειάζεται να εξετάσουμε προσεκτικά τον τρόπο λειτουργίας του extent tree ώστε να κατανοήσουμε πως μπορούμε να τροποποιήσουμε ή να κατακερματίσουμε τα προς ενημέρωση extent χωρίς να παρουσιαστούν ζητήματα συγχρονισμού. Όταν αναφερόμαστε σε κατακερματισμό, εννοούμε ότι αν κάνουμε εγγραφή σε κάποια συνεχόμενα cluster ενός μεγάλου extent, δεν θα θέλαμε να μεταφέρουμε ολόκληρο το extent στον τοπικό κόμβο προκειμένου να κάνουμε τις εγγραφές εκεί. Αντί αυτού, θα προτιμούσαμε να το διασπάσουμε σε (δύο ή τρία κατά περιπτώσεις) επιμέρους extent ώστε το ένα από αυτά να περιέχει τα cluster που θέλουμε να ενημερώσουμε και να μεταφέρουμε μόνο ό,τι είναι απαραίτητο.

Γίνεται αντιληπτό λοιπόν ότι η προτεινόμενη επέκταση παρουσιάζει προκλήσεις, αλλά μπορεί να είναι πολύ κερδοφόρα για μεγάλες ενημερώσεις αρχείων, ιδίως αν έχουμε λειτουργία write only στην οποία ίσως να μπορούμε να παρακάμψουμε την μεταφορά κάποιων σελίδων. Χρειάζεται όμως πρώτα να ποσοτικοποιηθεί το κόστος μεταφοράς σελίδων και διάσπασης του extent tree έναντι της απλής απομακρυσμένης πρόσβασης. Κατάλληλο σημείο έναρξης για τον πειραματισμό με τον κώδικα αποτελεί η δομή `if` κάτω από το σχόλιο "`Lookup extent status tree firstly`" της συνάρτησης `ext4_map_blocks`.

Κεφάλαιο 10

Παράρτημα

10.1 Διερεύνηση του φαινομένου "Read After Remote Write"

Σε αυτήν την ενότητα θα παρουσιαστεί η προσπάθεια που πραγματοποιήθηκε ώστε να προσδιοριστεί η αιτία του φαινομένου "Read After Remote Write." Παρόλο που δεν ήταν καταληκτική η διερεύνηση, αυτή παρατίθεται προκειμένου να αποτελέσει βάση προς συνέχιση σε κάποια μελλοντική στιγμή. Για ευκολία, περιγράφουμε ξανά το πρόβλημα:

Περιγραφή του προβλήματος

- Έχουμε πολλά απομακρυσμένα νήματα, το καθένα με το δικό του αρχείο και πρόσβαση μέσω DAX.
- Έχει προηγηθεί αμέσως πριν εγγραφή (στα ίδια ακριβώς αρχεία).
- Προσπαθούμε να κάνουμε ανάγνωση, τις πρώτες δύο φορές λαμβάνουμε σημαντικά μικρότερο bandwidth από το αναμενόμενο.
- Στην τρίτη εκτέλεση του πειράματος ανάγνωσης λαμβάνουμε φυσιολογική τιμή του bandwidth.

Προκειμένου να είμαστε βέβαιοι ότι για το φαινόμενο δεν ευθύνεται κάποια παράμετρος του FIO που δεν μπορούμε να γνωρίζουμε λόγω της πολυπλοκότητας του εργαλείου, δημιουργούμε μια πολύ απλοποιημένη μορφή του από την αρχή. Ονομάζουμε το απλοποιημένο εργαλείο cfio (custom FIO) και βρίσκεται στον κατάλογο `various/cfio.c` του πηγαίου κώδικα της εργασίας. Το πρόγραμμα έχει τις ακόλουθες παραμέτρους:

num_threads <int>	:	Αριθμός νημάτων (το καθένα επεξεργάζεται δικό του αρχείο)
directory <str>	:	Που τα νήματα βρίσκουν/δημιουργούν τα αρχεία
job_name <str>	:	Όνομα που έχουν/λαμβάνουν τα αρχεία (μορφής `directory/job_name_<thread_number>`)
file_size <long>	:	Μέγεθος αρχείων σε byte (χρήσιμο μόνο αν δεν υπάρχουν ήδη τα αρχεία και πρέπει να δημιουργηθούν)
block_size <int>	:	Μέγεθος buffer εγγραφής/ανάγνωσης σε bytes
loops <int>	:	Αριθμός επαναλήψεων περάσματος αρχείων
invalidate <bool>	:	Αφαίρεση αρχείων από την page cache πριν την εκτέλεση (Προεπιλογή: False)
is_write <bool>	:	Εγγραφή αρχείων (Προεπιλογή: Ανάγνωση αρχείων)

Ορίζουμε ένα σενάριο εκτέλεσης στο οποίο 16 απομακρυσμένα νήματα (με την έννοια ότι χρησιμοποιούμε τις συσκευές ενός μόνο NUMA κόμβου και τοποθετούμε τα νήματα στον άλλον) κάνουν πρώτα εγγραφή το καθένα στο δικό του αρχείο, και ύστερα διαβάζουν τα αρχεία τους τρεις φορές:

run_cfio.sh

```
#!/bin/bash

dir=$1
job=JOB
n=16
cpuset="16-31" # Remote access

# Write to the fileset
taskset -a -c $cpuset ./cfio --num_threads=$n --directory=$dir
    ↪ --job_name=$job --block_size=4096 --loops=1 --is_write

# Read the fileset
for i in {1..3}; do
        taskset -a -c $cpuset ./cfio --num_threads=$n
            ↪ --directory=$dir --job_name=$job
            ↪ --block_size=4096 --loops=1
done
```

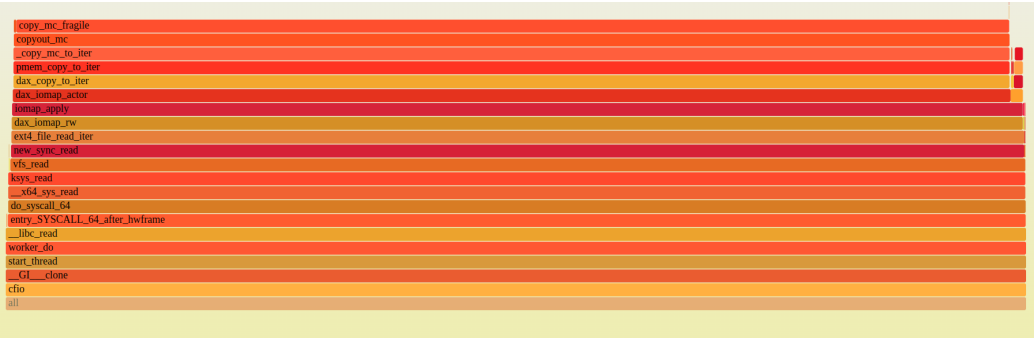
Λαμβάνουμε την ακόλουθη έξοδο:

```
$ ./run_cfio.sh /mnt/pmem0/phtof/daxdir
BW = 153,868 (Kbps)  # Write
BW = 1,694,802 (Kbps)  # Read 1
BW = 3,450,831 (Kbps)  # Read 2
BW = 14,765,409 (Kbps) # Read 3
```

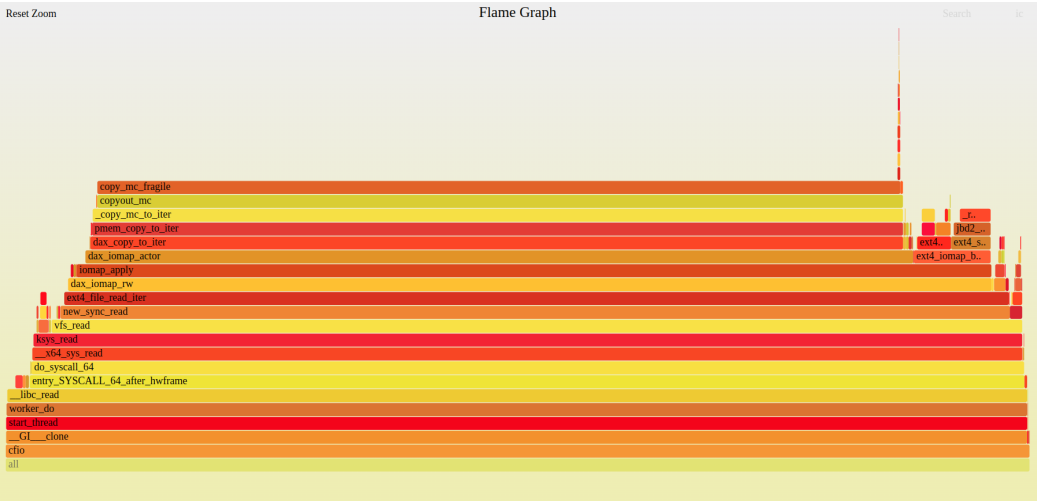
Βλέπουμε ότι χρειάζονται 3 εκτελέσεις για να λάβουμε μια λογική τιμή για την ρυθμαπόδοση ανάγνωσης. Εφόσον επιβεβαιώσαμε ότι το πρόβλημα δεν προκύπτει από την πολυπλοκότητα υλοποίησης του FIO, προχωράμε με την διερεύνηση.

10.1.1 Διερεύνηση της συμπεριφοράς του λειτουργικού συστήματος

Χρησιμοποιούμε flamegraph [7] προς ερμηνεία της καταγραφής του perf, ώστε να διαπιστώσουμε αν καλούνται ίσως κάποιες διαφορετικές συναρτήσεις μεταξύ προβληματικών και μη αναγνώσεων, λαμβάνοντας τα αποτελέσματα που φαίνονται στα σχήματα 10.1 και 10.2.



Σχήμα 10.1: Το Flamegraph που προκύπτει για την πρώτη σειρά αναγνώσεων



Σχήμα 10.2: Το Flamegraph που προκύπτει για την τρίτη σειρά αναγνώσεων

Αν και με πρώτη ματιά φαίνονται διαφορετικά τα δύο γραφήματα, στην πραγματικότητα έχουν αρκετά παρόμοια δείγματα. Στο πρώτο (που αντιστοιχεί στην πιο αργή εκτέλεση) οι συναρτήσεις που είναι χρονοβόρες (όπως η `copy_mc_fragile`) έχουν τόσα περισσότερα δείγματα που υποσκιάζουν δείγματα που σχετίζονται με συναρτήσεις προετοιμασίας για τις πιο ουσιώδεις συναρτήσεις. Αυτές οι συναρτήσεις προετοιμασίας είναι που φαίνονται πιο έντονα στο δεύτερο διάγραμμα (όπως η `ext4_iomap_begin`). Με προσεκτική εξέταση δηλαδή, δεν φαίνεται να διαφέρουν στο "ποιες συναρτήσεις εκτελούνται", αλλά στο "για πόσο εκτελούνται συγκεκριμένες συναρτήσεις."

Σε επόμενο στάδιο, πραγματοποιήθηκαν οι ίδιες μετρήσεις για τα συστήματα αρχείων NOVA και WineFS (σε αυτήν την εργασία θεωρούμε προσβάσεις στο ext4 όταν δεν αναφέρεται αλλιώς ρητά). Τα αποτελέσματα καταγράφονται στα αρχεία `experiments/measurements/nova_4G_read_after_write.json` και `experiments/measurements/winefs_4G_read_after_write.json`. Ωστόσο, η συμπεριφορά τους ήταν παρόμοια, παρόλο που τα συγκεκριμένα συστήματα αρχείων είναι ειδικά φτιαγμένα για αυξημένη επίδοση όταν εκτελούνται πάνω από μη πτητικές μνήμες.

Υποθέτουμε για τώρα ότι δεν συμβαίνει κάτι σε επίπεδο λειτουργικού το οποίο προκαλεί την διαφορά στις επιδόσεις των διαφορετικών σειρών αναγνώσεων.

10.1.2 Διερεύνηση της συμπεριφοράς των μη πτητικών συσκευών

Χρησιμοποιούμε το PMWatch για να λάβουμε στατιστικά για την χρήση της συσκευής κατά την εκτέλεση του προγράμματος. Αρχικά κάνουμε μια σειρά εγγραφών. Μετά αρχίζουμε το PMWatch με περίοδο 1 δευτερόλεπτο, το τρέχουμε στο background, αρχίζουμε την εκτέλεση μιας σειράς αναγνώσεων, περιμένουμε 1 δεύτερο για να έχουμε τουλάχιστον ένα δείγμα του PMWatch, και το σταματάμε.

```
$ ./run_cfio_once.sh 1 # Remote write
$ sudo pmwatch 1 > 1.txt & ./run_cfio_once.sh 0 ; sleep 1;
    ↪ sudo pmwatch-stop
$ sudo pmwatch 1 > 2.txt & ./run_cfio_once.sh 0 ; sleep 1;
    ↪ sudo pmwatch-stop
$ sudo pmwatch 1 > 3.txt & ./run_cfio_once.sh 0 ; sleep 1;
    ↪ sudo pmwatch-stop
```

Το `run_cfio_once.sh` είναι παρόμοιο του `run_cfio.sh`, με την διαφορά ότι το directory είναι προκαθορισμένο αντί για παράμετρος, και πλέον η μόνη παράμετρος του script αποτελεί μια δυαδική τιμή που είναι ίση με 0 για πραγ-

ματοποίηση αναγνώσεων, ή ίση με 1 για πραγματοποίηση εγγραφών.

Στο σχήμα 10.3 βλέπουμε μέρος των δεδομένων που λάβαμε για το πρώτο DIMM (εφόσον είναι interleaved τα DIMM ενός κόμβου, όποιο και να δούμε πιθανώς να δώσει αντιπροσωπευτική εικόνα). Παρατηρούμε ότι παρόλο που κάνουμε αναγνώσεις, η συσκευή δείχνει για την πρώτη σειρά αναγνώσεων ότι γίνονται και εγγραφές από τον επεξεργαστή (μάλιστα ίδιας τάξης μεγέθους σε bytes με τις εγγραφές). Αυτό παύει να ισχύει στην τρίτη σειρά αναγνώσεων, στην οποία φαίνεται να μην έχουμε καθόλου εγγραφές.

	A	B	C	D	E	F	G	H	I	J	K	L
1	timestamp		DIMM0									
2	epoch	timestamp	bytes_read (derived)	bytes_written (derived)	read_hit_ratio (derived)	write_hit_ratio (derived)	media_read_ops (derived)	media_write_ops (derived)	read_64B_ops_received	write_64B_ops_received	cpu_read_ops	cpu_write_ops
3	Εκτέλεση 1											
4	1677755101	2100104720	654775552	556669440	0.73	0.75	2557717	2174490	18928828	8697960	9310560	8629929
5	1677755102	2100103522	695417856	601703168	0.73	0.75	2716476	2350403	20267516	9401612	9972331	9328001
6	1677755103	2100102762	53248	799488	0	0	208	3123	13324	12492	38	17
7	Εκτέλεση 3											
8	1677755123	2100100714	53760	707072	0	0	210	2762	11888	11048	39	17
9	1677755124	2100101240	56832	802560	0	0	222	3135	13428	12540	37	17

Σχήμα 10.3: Μελέτη της πρώτης και της τρίτης σειράς αναγνώσεων μέσω του εργαλείου PMWatch της Intel

Έγινε δοκιμή για αναμονή ενός διαστήματος μεταξύ εγγραφών και της πρώτης σειράς αναγνώσεων σε περίπτωση που κάτι πραγματοποιούσαν εσωτερικά οι συσκευές μετά τις εγγραφές και αυτό επηρέαζε τις αναγνώσεις, αλλά το αποτέλεσμα είναι ακριβώς το ίδιο όση αναμονή και να υπάρξει μεταξύ εγγραφής και αναγνώσεων.

10.1.3 Διερεύνηση σε επίπεδο αρχιτεκτονικής του συστήματος

Βλέπουμε ότι ο επεξεργαστής στέλνει write operations στην συσκευή χωρίς να είναι αντιληπτό γιατί. Η πρώτη υπόθεση που έγινε είναι ότι ενδεχομένως αυτή η συμπεριφορά να σχετίζεται με τον iMC: Ίσως σε κάποια WPQ (ουρά αναμονής για εγγραφές) κάτι να συμβαίνει και να μην ολοκληρώνονται οι εγγραφές στην συσκευή, οπότε απλά κάθονται δεδομένα στο ADR χωρίς να έχουν γραφτεί στο μέσο. Ωστόσο, είναι δύσκολο να έχουμε εγγραφή τόσο μεγάλου όγκου δεδομένων στην πρώτη σειρά αναγνώσεων: Δεν φαίνεται κάποια δομή προσωρινής αποθήκευσης (buffer) στο path μεταξύ μέσου αποθήκευσης και επεξεργαστή να έχει τέτοια χωρητικότητα δεδομένων.

Στην διαδικασία αναζήτησης για να διαπιστωθεί αν αυτό το φαινόμενο έχει παρατηρηθεί και αλλού, βρέθηκε σχετικό έγγραφο [1] με μια συγκεντρωτική αναφορά προβλημάτων που έχει παρατηρήσει η Intel για τους επεξεργαστές Xeon 2ης γενιάς. Από τα προβλήματα που αναφέρονται, συγκρατήθηκαν με μια γρήγορη ανάγνωση οι επόμενοι τίτλοι που φάνηκαν σχετικοί. Σημειώνεται όμως ότι πολλοί από τους αναφερόμενους όρους δεν ήταν προφανείς, και για αυτό οι επιλογές μπορεί να είναι με μεγάλη πιθανότητα άστοχες. Ο μόνος

λόγος που σημειώνονται οι περισσότερες περιπτώσεις, είναι επειδή αναφέρουν κάποιους MSR, τους οποίους σε μια πιθανή επέκταση της διερεύνησης ίσως και να αξίζουν κάποια προσοχή.

Τα επόμενα δύο προβλήματα, αν και αναφέρονται στις μη πτητικές μνήμες, δεν μας αφορούν εφόσον οι μνήμες στα πειράματα μας ήταν ρυθμισμένες μόνο σε App Direct Mode.

- **Intel Optane Persistent Memory Mode or Mixed Mode May Cause a Hang**
  When the processor is utilizing Intel Optane Persistent Memory Mode or Mixed Mode, a Core MMIO read request may incorrectly block a write request from the IO subsystem, leading to a system hang.

- **Systems Using Intel Optane Persistent Memory in Mixed Mode May Experience a System Hang or Reset**
  When the processor is utilizing Intel Optane persistent memory in Mixed Mode with both App Direct and Memory Mode Snoopy Modes enabled, a system hang or reset may occur when running a workload.

Τα επόμενα τρία προβλήματα φαίνονται να είναι σχετικά, αλλά είτε αναφέρεται ότι οδηγούν σε hang και όχι απλή δυσλειτουργία, είτε αφορούν την λειτουργία ενός hardware prefetcher που δεν προσδιορίζεται υπό ποιες συνθήκες συμπεριφέρεται με μη αναμενόμενο τρόπο.

- **Memory Controller May Hang While in Virtual Lockstep**
  Under complex micro architectural conditions, a memory controller that is in Virtual Lockstep (VLS) may hang on a partial write transaction.

- **System May Hang When The Processor is in 3 Strike Due an Internal Mesh-to-mem Error**
  Under complex micro architectural conditions, the processor may hang with an error mesh-to-mem caused by a core 3-strike with Machine Check Exception (MSCOD=80h, MCACOD=0400h) logged into IA32_MC3_STATUS (MSR 40Dh).

- **XPT Prefetcher May Not Perform as Expected**
  When XPT prefetcher is enabled it may not prefetch as expected on memory channels that contain Intel Optane Persistent Memory.

Το επόμενο πρόβλημα είναι αυτό που θεωρώ ότι αξίζει κάποια διερεύνηση, καθώς φαίνεται το πιο σχετικό και δεν αναφέρει μόνο ενδεχόμενο system hang, αλλά και "unpredictable system behavior," που είναι ενθαρρυντικό. Εκτός αυτών, αναφέρει ρητά την περίπτωση που έχουμε παραπάνω από ένα socket

- **WBINVD/INVD Execution May Result in Unpredictable System Behavior**
  Under complex micro architectural conditions, the processor may hang or exhibit unpredictable system behavior during Writeback and Invalidate Cache (WBINVD) or Invalidate Internal Caches (INVD) cache instruction execution on a two or more socket system.

10.1.4 Πιθανή επέκταση της διερεύνησης

Σε κάποια προσπάθεια μελλοντικής επέκτασης της διερεύνησης που παρουσιάστηκε σε αυτήν την ενότητα, ορισμένες από τις πιθανές προσεγγίσεις είναι οι εξής:

- Ανάγνωση αρκετών hardware counters μέσω του perf stat και των σχετικών με pmem ονομασιών ή κωδικών που καταγράφονται από την intel [2]. Είναι αρκετά δύσκολη διαδικασία όμως, και πολλές φορές οι περιγραφές που δίνονται για τους counter δεν είναι ιδιαίτερα κατατοπιστικές.

- Πιθανή παρακολούθηση MSR registers, όπως μερικών από όσους αναφέρθηκαν στα προβλήματα που παρατέθηκαν προηγουμένως.

- Τρόπος για να ελέγξουμε αν αμέσως μετά τις εγγραφές έχει γίνει κανονικά το flush ή υπάρχουν εκκρεμότητες. Για παράδειγμα, θα φροντίζαμε να υπάρχουν αρχεία που περιέχουν ένα συγκεκριμένο αναγνωρίσιμο μοτίβο (π.χ. AAA...), να τροποποιήσουμε το cfio ώστε να γράφει ένα διαφορετικό, ευδιάκριτο του προαναφερθέντος μοτίβο (π.χ. BBB...) και να προβούμε στην εκτέλεση του για την εγγραφή. Ύστερα, θα θέλαμε να πραγματοποιήσουμε umount της συσκευής χωρίς να ενθαρρύνουμε περαιτέρω flushing, αν αυτό είναι εφικτό, ελέγχοντας τα περιεχόμενα του device για να αντιληφθούμε σε ποιον βαθμό έχει αντικατασταθεί το παλιό μοτίβο.

  Η μεθοδολογία αυτή παρουσιάζει αρκετές ιδιαιτερότητες που την κάνουν πιθανώς ανέφικτη, κυρίως γιατί η εντολή umount ή η ανάγνωση ενώ έχουμε προσαρτημένη την συσκευή είναι πολύ πιθανά να οδηγήσουν σε flush δεδομένων, και η παρατήρηση τελικά θα επηρεάσει το πείραμα.

10.2 Διερεύνηση της συμπεριφοράς του τελεστή ανακατεύθυνσης του Bash shell

Όπως αναφέρθηκε στο κύριο μέρος της εργασίας, οι τροποποιήσεις που πραγματοποιήσαμε στο ext4 δεν επιτυγχάνουν NUMA τοπικότητα όταν κάνουμε εγγραφές μέσω του τελεστή ανακατεύθυνσης του bash, ακόμα και αν έχουμε χρησιμοποιήσει taskset για να διασφαλίσουμε ότι οι εγγραφές γίνονται σε συγκεκριμένο NUMA κόμβο. Αυτό έχει ενδιαφέρον, γιατί ισοδύναμοι έλεγχοι (π.χ. μέσω του FIO) είναι επιτυχείς. Προς διερεύνηση του γεγονότος αυτού, επιστρατεύουμε την μέθοδο απομακρυσμένης αποσφαλμάτωσης (remote debugging) μιας εικονικής μηχανής του QEMU μέσω του GDB. Ορίζουμε το επόμενο breakpoint στο GDB:

```
break ext4_mb_new_blocks if (ar->inode->i_sb->s_mount_opt2 &
    ↪ 0x00000100)
```

Μέσω αυτού του breakpoint σταματάμε την εκτέλεση στην συνάρτηση `ext4_mb_new_blocks` μόνο όταν έχει δοθεί ως mount option η παράμετρος `numa` (που αντιστοιχεί στην μάσκα 0x00000100), ώστε να μην ενεργοποιείται επανειλημμένα το breakpoint λόγω ενεργειών της εικόνας ext4 του root directory. Εκτελώντας τον έλεγχο μέσω του αρχείου `experiments/scripts/numa/numa_test_redirection.sh`, ενεργοποιείται το breakpoint και στο GDB έχουμε την επόμενη εικόνα (στην οποία έχουμε απλοποιήσει την έξοδο για λόγους αναγνωσιμότητας):

```
(gdb) backtrace
#0 ext4_mb_new_blocks at fs/ext4/mballoc.c:5490
#1 0xffffffff812ce2f3 in ext4_ext_map_blocks ... at
    ↪ fs/ext4/extents.c:4246
#2 0xffffffff812e318e in ext4_map_blocks ... at
    ↪ fs/ext4/inode.c:638
#3 0xffffffff812e70f0 in mpage_map_one_extent ... at
    ↪ fs/ext4/inode.c:2395
#4 mpage_map_and_submit_extent ... at fs/ext4/inode.c:2448
#5 ext4_writepages at fs/ext4/inode.c:2800
#6 0xffffffff81193794 in do_writepages ... at
    ↪ mm/page-writeback.c:2352
#7 0xffffffff8124d3a7 in __writeback_single_inode ... at
    ↪ fs/fs-writeback.c:1467
#8 0xffffffff8124daab in writeback_sb_inodes ... at
    ↪ fs/fs-writeback.c:1732
#9 0xffffffff8124dedb in wb_writeback ... at
    ↪ fs/fs-writeback.c:1905
#10 0xffffffff8124e822 in wb_do_writeback ... at
    ↪ fs/fs-writeback.c:2050
#11 wb_workfn (work=...) at fs/fs-writeback.c:2091
```

```
#12 0xffffffff810887b4 in process_one_work (worker=... ,
    ↪ work=...) at kernel/workqueue.c:2276
#13 0xffffffff81088d4d in worker_thread (__worker=...) at
    ↪ kernel/workqueue.c:2422
#14 0xffffffff8108dc8b in kthread (_create=...) at
    ↪ kernel/kthread.c:313
#15 0xffffffff81001a12 in ret_from_fork () at
    ↪ arch/x86/entry/entry_64.S:294
(gdb) frame 13
(gdb) print worker->desc
\$10 = "flush-8:0", '\000' <repeats 14 times>
```

Επίσης, αξιοποιούμε το strace για να εξετάσουμε τον τρόπο λειτουργίας του τελεστή ανακατεύθυνση στο bash:

```
$ strace -r bash -c "echo AAAA > file"
```

Από την παραπάνω εντολή λαμβάνουμε το επόμενο μέρος της εξόδου της:

```
openat(AT_FDCWD, "file", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fcntl(1, F_GETFD) = 0
fcntl(1, F_DUPFD, 10) = 10
fcntl(1, F_GETFD) = 0
fcntl(10, F_SETFD, FD_CLOEXEC) = 0
dup2(3, 1) = 1
close(3) = 0
newfstatat(1, "", {st_mode=S_IFREG|0644, st_size=0, ...},
    ↪ AT_EMPTY_PATH) = 0
write(1, "AAAA\n", 5) = 5
```

Βλέπουμε λοιπόν ότι στην υλοποίηση του bash η ανακατεύθυνση φαίνεται να υλοποιείται μέσω μετάθεσης του ανοικτού αρχείου εγγραφής στο file descriptor του standard output, με χρήση της εντολής dup2. Με αυτόν τον τρόπο, υποθέτουμε ότι η εγγραφή στο ανοικτό αρχείο δεν γίνεται μέσω κάποιου syscall προς το VFS, αλλά εξαρτάται περισσότερα από την υλοποίηση του tty driver.

Ο driver πιθανώς να διατηρεί έναν buffer για την έξοδο της εντολής που επιθυμούμε να ανακατευθύνουμε, ο οποίος buffer υπόκειται flush κάποια στιγμή, βάσει της λογικής του memory management. Αυτό υποδεικνύουν συναρτήσεις τύπου `wb_do_writeback` στην έξοδο του backtrace. Το σημαντικό όμως είναι ότι αυτό πραγματοποιείται μέσω κάποιου kernel worker (`worker_thread` στην έξοδο του backtrace) ο οποίος αναλαμβάνει το flush των δεδομένων στο σύστημα αρχείων, όπως φαίνεται εξάλλου άμεσα από το description του worker.

Συμπερασματικά, βλέπουμε ότι στον έλεγχο με λογική ανακατεύθυνσης από την μια παρακάμπτεται πλήρως η λογική που αφορά το χαρακτηριστικό DAX (αφού πάντα περιλαμβάνεται η μνήμη στο μονοπάτι εγγραφών). Από την άλλη

ο κόμβος ο οποίος εν τέλει κάνει την εγγραφή δεν είναι αυτός του bash instance που πραγματοποιεί την ανακατεύθυνση, αλλά αυτός του kernel worker στον οποίον ανατίθεται η συγκεκριμένη δουλειά. Αν δεν υπάρχει kernel worker άμεσα διαθέσιμος για τον επιθυμητό κόμβο, τελικά το αρχείο θα καταλήξει αλλού από εκεί που θέλαμε.

10.3 Πιο λεπτομερής υποστήριξη DAX

Η λειτουργία DAX στο σύστημα αρχείων ext4 είναι ένα χαρακτηριστικό που έχει μεταμορφωθεί με πολλαπλούς τρόπους μεταξύ διαφορετικών εκδόσεων του πυρήνα Linux. Αρχικά, για παράδειγμα στην έκδοση 5.1, το σύστημα αρχείων υποστήριζε ενεργοποίηση της λειτουργίας DAX για όλα τα αρχεία του ή κανένα. Αργότερα, όπως φαίνεται σε εκδόσεις όπως η 5.10, η λειτουργία DAX ήταν κάτι που μπορούσε να ρυθμιστεί ανά αρχείο, αλλά για να επιβληθεί όντως η ρύθμιση σε ένα αρχείο έπρεπε αυτό να μην το αναφέρει κάποια διεργασία, και να μεσολαβήσει είτε drop-caches operation (ανανέωση των δεδομένων του συστήματος αρχείων στην DRAM), είτε να προσαρτηθεί εκ νέου το σύστημα αρχείων, είτε να γίνει επανεκκίνηση του συστήματος [8]. Σε πιο πρόσφατες εκδόσεις, όπως η 5.13, για να επιβληθεί η αλλαγή αρκεί μόνο το αρχείο να μην είναι ανοικτό από κάποια διεργασία [9].

Βλέπουμε λοιπόν ότι η τάση είναι να γίνεται όλο και πιο λεπτομερής η διαχείριση του χαρακτηριστικού DAX. Ιδανικά θα θέλαμε η ρύθμιση να εφαρμόζεται επιτόπου, χωρίς δηλαδή κάποια αυστηρή συνθήκη σχετικά με το αν είναι ανοικτό κάπου το αρχείο. Αυτό όμως δεν είναι άμεσα αντιληπτό πως μπορεί να υλοποιηθεί, καθώς απαιτεί πιο καθολική γνώση για τον σχεδιασμό του ext4, ειδικά σε επίπεδο συγχρονισμού.

Μπορούμε να φανταστούμε ότι αν έχουμε μετάβαση από DAX ενεργό σε ανενεργό για κάποιο αρχείο, αρκεί μόνο να επιτρέψουμε την αποθήκευση σελίδων στην page cache. Για το αντίστροφο, δηλαδή ενεργοποίηση του χαρακτηριστικού DAX για κάποιο αρχείο, η κατάσταση είναι πιο περίπλοκη γιατί απαιτείται κατάλληλο flush των σχετικών δεδομένων της page cache προτού αρχίσουμε να πραγματοποιούμε προσβάσεις σε αυτό άμεσα, την στιγμή μάλιστα που το αρχείο χρησιμοποιείται πιθανώς από πολλαπλές διεργασίες. Αυτό θα είχε άμεση ποινή απόδοσης για τις διεργασίες που έχουν ανοικτό το αρχείο, ιδίως αν η υλοποίηση μας επιχειρούσε να κάνει flush όλα τα δεδομένα του αρχείου στην page cache επιτόπου αντί κατά απαίτηση, δηλαδή flush κάποιων σελίδων την φορά όταν κάποια εφαρμογή αιτηθεί σχετική πρόσβαση. Αυτός ο σχεδιασμός βέβαια παρουσιάζει διαφορετικά προβλήματα. Η κατάσταση δυσκολεύει δραστικά αν λάβουμε κατά νου και την συμβατότητα με την διεπαφή `mmap`.

Η πρώτη περίπτωση που περιγράψαμε, δηλαδή η μετάβαση από DAX ενεργό σε ανενεργό, μελετήθηκε για τους σκοπούς αυτής της εργασίας. Οι σχετικές προσθήκες που έγιναν φαίνονται στις γραμμές `linux-5.13/fs/ext4/file.c:103-106` και `linux-5.13/fs/ext4/file.c:682-687` του πηγαίου κώδικα της εργασίας. Η συνθήκη της μετάβασης είναι απλή: αρκεί το αρχείο να έχει την δεδομένη στιγμή ενεργοποιημένο το χαρακτηριστικό DAX (inode flag `S_DAX`) και ταυτόχρονα κάποιος να έχει ζητήσει αλλαγή του χαρακτηριστικού αυτού (ext4 inode flag `EXT4_INODE_DAX`). Για λόγους πληρότητας, υποθέτουμε ότι κανονικά θα χρειαζόταν να ελέγξουμε επίσης ότι το αρχείο δεν χρησιμοποιείται ήδη από κάποια λειτουργία `mmap`. Για την ίδια την μετάβαση, αρκεί να ενημερώσουμε το `S_DAX` flag και ύστερα τα address space operations μέσω της συνάρτησης `ext4_set_aops`.

Ακόμα και σε αυτήν την απλή περίπτωση χρειάζεται προσοχή σε θέματα συγχρονισμού. Για παράδειγμα, στην συνάρτηση `ext4_file_write_iter` είναι αναγκαίο να κάνουμε τις αλλαγές κρατώντας το κλείδωμα του inode, εφόσον αυτό δεν έχει διεκδικηθεί ήδη όπως στην περίπτωση της συνάρτησης `ext4_dax_read_iter`. Οι αλλαγές αυτές έχουν ελεγχθεί με συνοπτικές διαδικασίες, αλλά φαίνονται να είναι ευσταθείς (κάτι το οποίο δεν ισχύει αν δεν ενημερώσουμε τα address space operations) και ορθές στον βαθμό που πράγματι παρατηρείται η μετάβαση όσο το αρχείο είναι ανοικτό.

Σίγουρα θα είχε μεγαλύτερο ενδιαφέρον να επιχειρήσουμε την υλοποίηση της αντίστροφης μετάβασης, αλλά αυτό ξέφευγε κατά πολύ του διαθέσιμου χρόνου εκπόνησης της παρούσας εργασίας. Αν είχαμε υλοποίηση και των δύο κατευθύνσεων, θα επιτρεπόταν η ουσιαστική μοντελοποίηση ενός συστήματος στο οποίο θα αξιοποιούταν τόσο η λειτουργία DAX όσο και η page cache για αποδοτικότερη διαχείριση του αθροιστικού bandwidth πτητικών και μη συσκευών όταν υπάρχουν αρχεία με μεγάλη επαναληψιμότητα στις προσβάσεις τους.

Βιβλιογραφία

[1] *2nd Gen Intel®Xeon®Scalable Processors: Specification Update April 2023*. URL: https://cdrdv2-public.intel.com/338848/338848_2nd%20Gen%20Intel%C2%AE%20Xeon%C2%AE%20Scalable%20Processors%20Specification%20Update_Rev027US.pdf (Ημερομηνία Τελευταίας Πρόσβασης: 10/07/2023).

[2] *2nd Generation Intel®Xeon®Processor Scalable Family based on Cascade Lake product: Reference for hardware events that can be monitored for the CPU(s)*. URL: https://perfmon-events.intel.com/cascadelake_server.html.

[3] *3D XPoint Wikipedia article*. URL: https://en.wikipedia.org/wiki/3D_XPoint.

[4] *3D XPoint™: A Breakthrough in Non-Volatile Memory Technology*. URL: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-micron-3d-xpoint-webcast.html.

[5] Brian Beeler. *Intel Optane DC Persistent Memory Module (PMM)*. URL: https://www.storagereview.com/news/intel-optane-dc-persistent-memory-module-pmm.

[6] *Bigalloc Documentation*. URL: https://ext4.wiki.kernel.org/index.php/Bigalloc.

[7] *CPU FLame Graphs*. URL: https://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html.

[8] *Dax documentation before v5.13*. URL: https://github.com/torvalds/linux/blob/v5.10/Documentation/filesystems/dax.txt.

[9] *Dax documentation from v5.13 and afterwards*. URL: https://github.com/torvalds/linux/blob/v5.13/Documentation/filesystems/dax.txt.

[10] *Device Mapper Administrator's Guide*. URL: https://docs.kernel.org/admin-guide/device-mapper/.

[11] *FIO documentation*. URL: https://fio.readthedocs.io/en/latest/fio_doc.html.

[12] *General Information on the Device Mapper*. URL: https://en.wikipedia.org/wiki/Device%5C_mapper.

[13] Jungwook Han κ.ά. «Is Data Migration Evil in the NVM File System?» Στο: *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE. 2021, σσ. 26–31.

[14] *Hawkeye: GitHub repository*. URL: https://github.com/apanwariisc/x86-MMU-Profiler.

[15] *IPMCTL Documentation: Creation of memory allocation goal*. URL: https://docs.pmem.io/ipmctl-user-guide/provisioning/create-memory-allocation-goal.

[16] Joseph Izraelevitz κ.ά. «Basic performance measurements of the intel optane DC persistent memory module». Στο: *arXiv preprint arXiv:1903.05714* (2019).

[17] Wenqing Jia, Dejun Jiang και Jin Xiong. «NapFS: A High-Performance NUMA-Aware PM File System». Στο: *2022 IEEE 40th International Conference on Computer Design (ICCD)*. 2022, σσ. 593–601. DOI: 10.1109/ICCD56317.2022.00093.

[18] Rohan Kadekodi κ.ά. «WineFS: a hugepage-aware file system for persistent memory that ages gracefully». Στο: *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 2021, σσ. 804–818.

[19] Saurabh Kadekodi, Vaishnavh Nagarajan και Gregory R Ganger. «Geriatrix: Aging what you see and what you {don't} see. A file system aging approach for modern storage systems». Στο: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 2018, σσ. 691–704.

[20] *Non-Uniform Memory Access Architecture Wikipedia article*. URL: https://en.wikipedia.org/wiki/Non-uniform_memory_access.

[21] *NOVA extension for supporting multiple NUMA nodes*. URL: https://github.com/utsaslab/WineFS/tree/main/Linux-5.1/fs/nnova.

[22] *Open Virtual Machine Firmware*. URL: https://wiki.ubuntu.com/UEFI/OVMF.

[23] Ashish Panwar, Sorav Bansal και K Gopinath. «Hawkeye: Efficient fine-grained os support for huge pages». Στο: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019, σσ. 347–360.

[24] *Persistent Memory Development Kit*. URL: https://pmem.io/pmdk/.

[25] *proc(5) linux manual page*. URL: https://man7.org/linux/man-pages/man5/proc.5.html.

[26] *The Linux Function Tracer*. URL: https://www.kernel.org/doc/html/v5.0/trace/ftrace.html.

[27] Erez Zadok Vasily Tarasov και Spencer Shepler. *Filebench: A Flexible Framework for File System Benchmarking*. URL: https://www.usenix.org/system/files/login/articles/login_spring16_02_tarasov.pdf.

[28] Ying Wang, Dejun Jiang και Jin Xiong. «Numa-aware thread migration for high performance nvmm file systems». Στο: *Proceedings of the 36th International Conference on Massive Storage Systems and Technology*. 2020.

[29] Jian Xu και Steven Swanson. «{NOVA}: A log-structured file system for hybrid {Volatile/Non-volatile} main memories». Στο: *14th USENIX Conference on File and Storage Technologies (FAST 16)*. 2016, σσ. 323–338.

[30] Jian Yang κ.ά. «An empirical guide to the behavior and use of scalable persistent memory». Στο: *18th USENIX Conference on File and Storage Technologies (FAST 20)*. 2020, σσ. 169–182.