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

Υλοποίηση μηχανισμού υπολογισμού Miss Ratio καμπυλών σε περιβάλλοντα διαχωρισμένης μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΑΝΑΓΟΠΟΥΛΟΥ ΑΝΝΑ

Επιβλέπων : Γεώργιος Γκούμας
Επίκουρος Καθηγητής

Αθήνα, Οκτώβριος 2020

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

Υλοποίηση μηχανισμού υπολογισμού Miss Ratio καμπυλών σε περιβάλλοντα διαχωρισμένης μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΑΝΑΓΟΠΟΥΛΟΥ ΑΝΝΑ

Επιβλέπων : Γεώργιος Γκούμας
Επίκουρος Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22η Οκτωβρίου 2020.

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

Αθήνα, Οκτώβριος 2020

Παναγοπούλου Άννα

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

Περίληψη

Αντικείμενο της διπλωματικής εργασίας αποτελεί η μελέτη του μοντέλου Memory Disaggregation (μοντέλο διαχωρισμένης μνήμης) ως μια υποσχόμενη προσέγγιση για καταλληλότερη διαχείριση των πόρων μνήμης στο Cloud. Συγκεκριμένα, επεκτείναμε το σύστημα FluidMem με την ανάπτυξη ενός αλγορίθμου που υπολογίζει αποστάσεις επαναχρησιμοποίησης σε σελίδες διεργασιών ή εικονικών μηχανημάτων για προσβάσεις που ανιχνεύει ο μηχανισμός σελιδοποίησης του FluidMem. Ο μηχανισμός σελιδοποίησης που παρέχει το FluidMem κρίθηκε ως κατάλληλη διεπαφή επί της οποίας μπορεί να ληφθεί η προαναφερθείσα μετρική διότι είναι υλοποιημένος σε επίπεδο χρήστη και συνεπώς δεν απαιτούνται τροποποιήσεις στον πυρήνα του λειτουργικού συστήματος. Οι αποστάσεις επαναχρησιμοποίησης λαμβάνονται έπειτα από αίτημα για έναρξη της λήψης του αποτυπώματος μνήμης για μια διεργασία που επιβλέπει το FluidMem και αφορά όλες τις προσβάσεις σε σελίδες της διεργασίας που ανιχνεύει το FluidMem από την απαρχή του αιτήματος έως το ακόλουθο αίτημα τέλους. Τα αποτελέσματα του αποτυπώματος της μνήμης της διεργασίας κατά το συγκεκριμένο χρονικό διάστημα μπορούν έπειτα να διέλθουν από ένα στάδιο επεξεργασίας από το οποίο θα προκύψουν (i) το Ιστόγραμμα Αποστάσεων Επαναχρησιμοποίησης (Reuse Distance Histogram) και (ii) η Miss Ratio Καμπύλη (MRC) της διεργασίας.

Η μέθοδος που εφαρμόσαμε, συνιστά έναν non-intrusive τρόπο που μας παρέχει ικανότητα εκτίμησης της Ενεργά Χρησιμοποιούμενης Μνήμης (WSS) για μεγέθη που μπορούν να είναι και μικρότερα της αρχικής μνήμης που εκχωρείται στα εικονικά μηχανήματα. Το ελάχιστο μέγεθος του WSS που είμαστε σε θέση να ανιχνεύσουμε, εξαρτάται αποκλειστικά και μόνο από το μέγεθος του resizable LRU Buffer του FluidMem, και δεν έχει σχέση με το μέγεθος της μνήμης που αποδόθηκε στο εικονικό μηχάνημα κατά την έναρξή του. Επίσης, μας επιτρέπει να λάβουμε αποφάσεις κατάλληλης αυξομείωσης του μεγέθους του LRU Buffer, αλλά και να προσθέτουμε απομακρυσμένη μνήμη στα εικονικά μηχανήματα, όπου υπάρχει ανάγκη.

Ο αλγόριθμος που προτείνεται αποτελεί μια κομψή υλοποίηση του αλγορίθμου αναζήτησης δέντρων του Olken με αξιοποίηση AVL δέντρων για την αναπαράσταση της στοίβας. Εισάγει πολυπλοκότητα $O(NlogM)$, δεδομένου ότι N είναι ο αριθμός των συνολικών προσβάσεων σε σελίδες που ανιχνεύει το FluidMem και οι μοναδικές προσβάσεις.

Επίσης, το παρατηρούμενο overhead του αλγορίθμου είναι της τάξης του 15%, και θεωρείται αρκετά μικρό ώστε να αντισταθμίζεται από τα οφέλη που προκύπτουν κατά τη γνώση του WSS των διεργασιών.

Λέξεις κλειδιά

Μοντέλο διαχωρισμένης μνήμης, FluidMem, διεργασία, εικονικά μηχανήματα, μηχανισμός σελιδοποίησης, επίπεδο χρήστη, αποτύπωμα μνήμης, Ιστόγραμμα Αποστάσεων Επαναχρησιμοποίησης, MRC, αλγόριθμος αναζήτησης δέντρων του Olken, AVL δέντρα

Abstract

The purpose of this diploma thesis is to extend FluidMem system with a utility that computes the reuse distance of every page-faulted address trapped by FluidMem's pagefault handler. The paging mechanism that comes with FluidMem is considered to be a convenient interface for embedding the aforementioned metric mainly because it is implemented on userspace and thus no modifications to the kernelspace layer are required.

The reuse distance computation for the pages of a process monitored by FluidMem is provided upon user request and it incorporates every page access detected by FluidMem from the onset of the request till a subsequent request to stop. The results of the profiling operation during this period can be then further processed to obtain the Reuse Distance Histogram and the Miss Ratio Curve of the process.

The method we introduce is a non-intrusive way that provides us with the capability to determine the actual Working Set Size (WSS) of VMs, for sizes that can be less than the initial memory given to VMs on boot time. In fact, the least size of the WSS we can detect, depends only on the capacity of FluidMem's resizable LRU Buffer. Due to this method, we are capable of making considerate decisions regarding sizing up or down the LRU Buffer capacity, as well as increasing the remote memory given to VMs, in cases they are in need.

This thesis recommends an algorithmic approach that implements Olken search trees algorithm using AVL trees. It introduces worst-time complexity of $O(NlogM)$, where N is the overall number and M is the unique appearence, of page accesses FluidMem has detected as pagefaults for a process.

Our results reveal an overhead that is approximately 15%, which is low enough considering the benefits we obtain by knowing the WSS of VMs.

Key words

Memory Disaggregation, FluidMem, process, VMs, WSS, userspace pagefault handler, Reuse Distance Histogram, Miss Ratio Curve, Olken search trees algorithm, AVL trees

Ευχαριστίες

Ευχαριστώ ιδιαίτερα τον καθηγητή μου κ.Γεώργιο Γκούμα για τη δυνατότητα που μου έδωσε να αναλάβω τη συγκεκριμένη εργασία, και που με δέχτηκε να υλοποιήσω ένα κομμάτι έρευνας στο εργαστήριό του.

Ευχαριστώ πολύ τον μεταδιδακτορικό ερευνητή Δρ. Στέφανο Γεράνγκελο για την αρχική ιδέα ενασχόλησης με το θέμα και τις εναρκτήριες συμβουλές και κατευθύνσεις που μου έδωσε.

Ευχαριστώ πολύ τον διδακτορικό ερευνητή κ. Στράτο Ψωμαδάκη για όλες τις υποδείξεις και τις κατευθυντήριες γραμμές που μου έδωσε κατά την πορεία της εργασίας καθώς και για τις διορθώσεις που μου έκανε κατά το τέλος της.

Επίσης, θα ήθελα να ευχαριστήσω τους καθηγητές μου και κυρίως τους κ. Νεκτάριο Κοζύρη και κ. Γεώργιο Γκούμα, των οποίων οι διαλέξεις μου μετέδωσαν μέρος του πηγαίου ενδιαφέροντός τους για τον τομέα των υπολογιστικών συστημάτων, οδηγώντας με στην εκπόνηση της διπλωματικής μου εργασίας στον τομέα αυτό.

Τέλος, θα ήθελα να πω ένα ευχαριστώ στους γονείς μου, τον αδερφό μου και τον σύντροφό μου, που ο καθένας ξεχωριστά με τον δικό του τρόπο, ήταν δίπλα μου σε κάθε δυσκολία και μου έδινε τη δύναμη να κυνηγήσω τα όνειρά μου.

Παναγοπούλου Άννα,

Αθήνα, 22η Οκτωβρίου 2020

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

Στο παρόν κεφάλαιο παρουσιάζεται ο σκοπός της διπλωματικής εργασίας καθώς και τα κίνητρα που οδήγησαν στην εκπόνησή της. Έπειτα, γίνεται αναφορά στην οργάνωση της διπλωματικής εργασίας και παρατίθενται επιγραμματικά τα περιεχόμενα των επιμέρους κεφαλαίων.

1.1 Σκοπός της Διπλωματικής Εργασίας

Σύμφωνα με σύγχρονες έρευνες στα υπολογιστικά συστήματα, το Memory Disaggregation μοντέλο (διαχωρισμού ή αποσύνδεσης της μνήμης) [1], [2], [3] συνιστά μια πολλά υποσχόμενη προσέγγιση, τόσο σε ακαδημαϊκό όσο και σε πρακτικό επίπεδο στην δρομολόγηση προβλημάτων ανισορροπιών που αφορούν στη χρήση μνήμης. Σε περιβάλλοντα εικονικοποίησης στο Cloud οι διακυμάνσεις σε χρήση μνήμης αποτελούν ένα πολύ εμφανές και σοβαρό πρόβλημα που οδηγεί σε φαινόμενα υπερβολικού paging και thrashing, ακόμη και αν υπάρχει επαρκή ποσότητα μη χρησιμοποιούμενης μνήμης στον ίδιο υπολογιστικό κόμβο ή σε άλλους κόμβους του cluster (σύμπλεγμα) [1].

Ένα από τα συστήματα που υλοποιήθηκαν με στόχο την απόδειξη ότι το Full Memory Disaggregation μοντέλο (μοντέλο πλήρους διαχωρισμού ή αποσύνδεσης της μνήμης) μπορεί πράγματι να ενσωματωθεί στα σημερινά κέντρα δεδομένων (datacenters), χωρίς την εισαγωγή επιπρόσθετων απαιτήσεων που αφορούν σε χρήση εξειδικευμένου υλικού ή τροποποιήσεις σε επίπεδο λειτουργικού συστήματος είναι το FluidMem [4]. Η συνεισφορά του FluidMem στην επιστημονική κοινότητα, σχετίζεται με την προσφορά της δυνατότητας οποιαδήποτε σελίδα ενός εικονικού μηχανήματος να είναι εφικτό να βρίσκεται σε κάποιον διαφορετικό υπολογιστικό κόμβο στο Cloud cluster. Από εκεί και πέρα, η ιδιότητα αυτή μπορεί να αξιοποιηθεί με ποικίλους τρόπους για τη λήψη εύλογων αποφάσεων διανομής απομακρυσμένων σελίδων μνήμης σε εικονικά μηχανήματα που έχουν ανάγκη, αλλά και αποφάσεων ανάκλησης σελίδων από αυτά, όταν δεν γίνεται ενεργή χρησιμοποίησή τους.

Οι διαχειριστικές αποφάσεις κατά τη χρήση μνήμης στο Cloud, σε ένα πλαίσιο όπου καθίσταται εφικτή η πλήρης αποσύνδεση των σελίδων των εικονικών μηχανημάτων από τον εκάστοτε υπολογιστικό κόμβο στον οποίο ανήκουν είναι το ευρύτερο αντικείμενο μελέτης της παρούσας διπλωματική. Συγκεκριμένα, ο σκοπός της εργασίας είναι η επέκταση του συστήματος FluidMem που θα μας φέρει ένα βήμα πλησιέστερα στη λήψη ορθών αποφάσεων διανομής της μνήμης σε ένα fully-disaggregated σύστημα.

Πλήθος από μελέτες αποδεικνύουν ότι μια από τις κυριότερες μετρικές που καθορίζει την ανάγκη σε μνήμη των εικονικών μηχανημάτων είναι η ενεργά χρησιμοποιούμενη μνήμη, ή αλλιώς το μέγεθος του συνόλου εργασίας τους (WSS). Στα πλαίσια της παρούσας διπλωματικής, μελετάται η ένταξη της εκτίμησης της ενεργά χρησιμοποιούμενης μνήμης των εικονικών μηχανημάτων στον μηχανισμό σελιδοποίησης του FluidMem. Η εκτίμηση του μεγέθους της ενεργά χρησιμοποιούμενης μνήμης γίνεται με την κατάστρωση MRC καμπυλών, όπως προκύπτει από τη λήψη μετρικών επαναχρησιμοποίησης (reuse distances) σελίδων σε προσβάσεις που ανιχνεύονται από τον μηχανισμό σελιδοποίησης.

Η συγκεκριμένη μέθοδος παρέχει τελικά μια εικόνα για το ποσό της μνήμης που κάθε εικονικό μηχάνημα ιδανικά επιθυμεί να του αποδοθεί, με στόχο την ελαχιστοποίηση του λόγου απώλειας προσβάσεων (miss ratio). Πρόκειται, επίσης, για μια μέθοδο που δεν απαιτεί τροποποιήσεις σε επίπεδο πυρήνα και είναι υλοποιημένο σε επίπεδο χρήστη (userspace). Επομένως η μοναδική επέμβαση που

απαιτεί για να εισαχθεί η γνώση του μεγέθους συνόλου εργασίας των εικονικών μηχανημάτων σε ένα fully-disaggregated σύστημα στο Cloud που κάνει χρήση του συστήματος FluidMem, είναι ελάχιστες αλλαγές στο ίδιο το FluidMem.

1.2 Οργάνωση του Τόμου

Τα επόμενα κεφάλαια οργανώνονται ως εξής:

Στο κεφάλαιο 2 γίνεται παράθεση του θεωρητικού υποβάθρου και της σχετικής υπάρχουσας βιβλιογραφίας που αφορά στην ενεργά χρησιμοποιούμενη μνήμη και τις MRC καμπύλες. Με αφετηρία την εικονικοποίηση και τις υπάρχουσες τεχνολογίες επιβλεπόντων στη συνέχεια δίνεται μια εικόνα της χρησιμότητας γνώσης της ενεργά χρησιμοποιούμενης μνήμης και του υπολογισμού της με τη βοήθεια των MRC καμπυλών. Στη συνέχεια, παρατίθεται το θεωρητικό υπόβαθρο που αφορά στις αλγοριθμικές προσεγγίσεις για την παραγωγή MRC καμπυλών. Τέλος, αναλύονται υπάρχοντα συστήματα που καταστρώνουν MRC καμπύλες με πλήθος τρόπων.

Στο κεφάλαιο 3 παρουσιάζεται το σύστημα FluidMem με τα βασικά δομικά του μέρη. Επίσης, αναλύεται το RAMCloud key-value σύστημα αποθήκευσης, που δίνει και τα καλύτερα αποτελέσματα όταν χρησιμοποιείται για το ρόλο της απομακρυσμένης μνήμης. Το κεφάλαιο 3 ολοκληρώνεται με ανάλυση της συνεισφοράς της παρούσας διπλωματικής εργασίας.

Στο κεφάλαιο 4 δίνονται οι λεπτομέρειες της υλοποίησης για την ένταξη του υπολογισμού MRC καμπυλών στο FluidMem.

Το κεφάλαιο 5 επικεντρώνεται στην παρουσίαση των πειραμάτων που υλοποιήσαμε και την αξιολόγησή τους. Επίσης, παρατίθεται τμήμα με τεχνικά θέματα που παρατηρήθηκαν στο FluidMem κατά τη διενέργεια των πειραμάτων. Το κεφάλαιο 6 κλείνει την εργασία με σύνοψη των συμπερασμάτων που προέκυψαν κατά την αξιολόγηση και με παράθεση πιθανών μελλοντικών επεκτάσεων της εργασίας.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

2.1 Εικονικοποίηση

Στις ημέρες μας, η υπολογιστική σύννεφου αποτελεί μια από τις επικρατέστερες τάσεις όσο η τεχνολογία παρέχει τη δυνατότητα σε χρήστες να έχουν πρόσβαση σε πολύ μεγάλο όγκο δεδομένων, πληροφορίας και πληθώρας άλλων υπολογιστικών πόρων. Η εικονικοποίηση (virtualization) είναι ένα είδος τεχνικής που επιφέρει τη διαθεσιμότητα της υπολογιστικής σύννεφου. Πρόκειται για μια μέθοδο που επιτρέπει τη δημιουργία ενός αφηρημένου στρώματος από υπολογιστικούς πόρους σύννεφου αποκρύπτοντας παράλληλα την πολυπλοκότητα του λογισμικού και του υλικού των συστημάτων. Τα τελευταία χρόνια, έχει αναδειχθεί σε μια πολλά υποσχόμενη λύση στην ελαχιστοποίηση του κόστους που απαιτείται για τη λειτουργία των υπολογιστικών κόμβων σε έναν πάροχο [5].

Η ιδέα της εικονικοποίησης είχε αρχικά παρουσιαστεί από την IBM τη δεκαετία του 1960, η οποία περιέγραψε τη δυνατότητα πλήθος από διαφορετικά λειτουργικά συστήματα να εγκαθίστανται σε έναν μοναδικό υπολογιστή [6]. Ειδικότερα, η εικονικοποίηση είναι μια μέθοδος που δημιουργεί εικονικά μηχανήματα με προσομοίωση των φυσικών πόρων ενός κόμβου. Καθιστά, έτσι, εφικτό πολλά εικονικά μηχανήματα να είναι συγχρόνως ενεργά στον ίδιο κόμβο, την ίδια χρονική στιγμή. Η ιδιότητα αυτή επιτρέπει σε φορτία που πρέπει να κατανεμηθούν σε πλήθος από κόμβους να ανατίθενται σε πολλά εικονικά μηχανήματα σε έναν μόνο κόμβο. Συνεπώς, οδηγούμαστε σε πιο αποδοτική χρήση των πόρων μειώνοντας το πλήθος των κόμβων που πρέπει να είναι ενεργά κάθε χρονική στιγμή [7].

Τελικά, η εικονικοποίηση οδηγεί στην μείωση χρήσης του υλικού, στη μείωση του συνολικού κόστους και στην εξοικονόμηση ενέργειας, διευκολύνοντας παράλληλα τη συνύπαρξη πολλών εφαρμογών και λειτουργικών συστημάτων στον ίδιο κόμβο την ίδια στιγμή. Συνεπώς, βελτιώνει την απόδοση, την ευελιξία και την χρήση του υποκείμενου υλικού των υπολογιστικών συστημάτων [5].

2.1.1 Τεχνικές εικονικοποίησης λογισμικού

Οι τεχνικές εικονικοποίησης λογισμικού μπορούν να ομαδοποιηθούν σε τρεις κύριες κατηγορίες [6]:

- **Πλήρης εικονικοποίηση**
  Πρόκειται για μια προσέγγιση δημιουργίας εικονικού περιβάλλοντος επί του οποίου μπορούν να εκτελεστούν μη-τροποποιημένες εικόνες λειτουργικών συστημάτων. Προσφέρει πλήρη αναπαραγωγή της πρωτότυπης συμπεριφοράς του φιλοξενούμενου λειτουργικού συστήματος και διευκολύνσεις στο λειτουργικό σύστημα του οικοδεσπότη (host).

- **Containers**
  Πρόκειται για μια προσέγγιση που βασίζεται στον πυρήνα ενός μοναδικού λειτουργικού συστήματος, ενισχυμένου με οριοθετήσεις που προσφέρουν αυξημένη απομόνωση μεταξύ ομάδων διεργασιών. Ειδικότερα, τα containers παρέχουν τη δυνατότητα εκτέλεσης πλήθους από στιγμιότυπα εικονικοποιημένων λειτουργικών συστημάτων σε ένα μοναδικό στιγμιότυπο του πραγματικού εικονικού συστήματος.

- **Παρα-εικονικοποίηση**
  Πρόκειται για μια προσέγγιση που διαχειρίζεται τα προβλήματα απόδοσης που είναι υπαρκτά

σε ένα τυπικό σύστημα πλήρους εικονικοποίησης χωρίς προσπάθεια ακριβούς αναπαραγωγής της πρωτότυπης συμπεριφοράς του φιλοξενούμενου λειτουργικού συστήματος. Η συγκεκριμένη προσέγγιση απαιτεί τροποποιήσεις στο φιλοξενούμενο λειτουργικό σύστημα ώστε να λειτουργήσει στο παρα-εικονικοποιημένο περιβάλλον. Οι αλλαγές σχετίζονται με την αναδρομολόγηση λειτουργιών που αφορούν στην εικονικοποίηση απευθείας στον επιβλέποντα, αντί να διέλθουν πρώτα από το λειτουργικό σύστημα όπως συμβαίνει σε περιπτώσεις απλής εικονικοποίησης. Ένα από τα σημαντικά μειονεκτήματα της πλήρους εικονικοποίησης που επιλύει, είναι περιπτώσεις κατά τις οποίες είναι επιθυμητό για τα φιλοξενούμενα συστήματα να μπορούν να αντιλαμβάνονται τόσο πραγματικούς όσο και εικονικούς πόρους, ώστε το λειτουργικό σύστημα να υποστηρίζει καλύτερα χρονικά ευαίσθητα καθήκοντα. Ο επιβλέποντας Xen [8] στηρίζεται στην παραεικονικοποίηση, χωρίς παράλληλα να απαιτεί τροποποιήσεις σε επίπεδο εφαρμογών των φιλοξενούμενων μηχανημάτων.

2.1.2 Επιβλέποντας

Οι πλατφόρμες εικονικοποίησης σε επίπεδο λογισμικού δύνανται να υποστηρίξουν πλήθος από εικονικά συστήματα στον ίδιο επεξεργαστή με τρόπο τέτοιον ώστε να λειτουργούν απομονωμένα το ένα από το άλλο [6]. Σε μια πλατφόρμα εικονικοποίησης επιπέδου λογισμικού, η εικονικοποίηση εφαρμόζεται με την τεχνολογία του επιβλέποντα (hypervisor ή, εναλλακτικά, του ελεγκτή των εικονικών μηχανημάτων - VMM), δηλαδή στοιχεία λογισμικού ή firmware που εικονικοποιούν τους πόρους τους συστήματος.

Ο επιβλέποντας είναι διεπαφή που βρίσκεται ανάμεσα στο υλικό και το λειτουργικό σύστημα του φιλοξενούμενου εικονικού μηχανήματος. Σε ότι αφορά στη λειτουργικότητά του, στην πραγματικότητα ελέγχει τη μνήμη, τη σύνδεση δικτύου και άλλων στοιχείων επιτρέποντας πλήθος από λειτουργικά συστήματα να συνυπάρχουν σε έναν μόνο κόμβο χωρίς τη βοήθεια πηγαίου κώδικα. Άλλωστε, το λειτουργικό σύστημα των φιλοξενούμενων εικονικών μηχανημάτων από μόνο του δεν είναι ικανό να διαχειριστεί εργασίες που αφορούν ένα σύνολο από κόμβους του cluster.

Υπάρχουν δύο κατηγορίες επιβλεπόντων:

- **Bare Metal/ Native επιβλέποντας ή Τύπου 1**
  Πρόκειται για επιβλέποντες που εντοπίζονται απευθείας στο υλικό και προσφέρουν επικοινωνία ανάμεσα στα εικονικά μηχανήματα και το υλικό. Ακριβώς επειδή αυτή η κατηγορία λειτουργεί απευθείας στο υλικό, δεν υπάρχει καμία απαίτηση ως προς το λειτουργικό σύστημα του χρήστη. Επιγραμματικά μερικά παραδείγματα επιβλεπόντων τύπου 1 είναι: Citrix XenServer, Microsoft Windows Server 2012 Hyper-V VMware vSphere/ESXi, ανοιχτού κώδικα Kernel-based Virtual Machine (KVM) και RedHat Enterprise Virtualization (RHEV). [5]

- **Φιλοξενούμενος επιβλέποντας ή Τύπου 2**
  Πρόκειται για επιβλέποντες που εντοπίζονται στο λειτουργικό σύστημα και διαχειρίζονται από εκεί τα εικονικά μηχανήματα. Στον φιλοξενούμενο επιβλέποντα υπάρχει ένα επιπλέον επίπεδο ανάμεσα στα φυσικά και τα εικονικά μηχανήματα, που παρέχει στο φιλοξενούμενο λειτουργικό σύστημα την απαραίτητη ασφάλεια και απομόνωση. Μερικά παραδείγματα επιβλεπόντων τύπου 2: Microsoft Virtual PC, VirtualBox, Vmware Workstation και Oracle Virtual Box [5]

2.2 Ενεργά Χρησιμοποιούμενη Μνήμη και MRC καμπύλες

Η εικονικοποίηση, προκειμένου να εφαρμοστεί σωστά από άποψη διαχείρισης, επιφέρει μια σειρά από προκλήσεις. Συγκεκριμένα, η ταυτόχρονη συνύπαρξη των εικονικών μηχανημάτων σημαίνει και ταυτόχρονη ζήτηση για ίδιους υπολογιστικούς πόρους, όπως η μνήμη και ο δίσκος. Συνεπώς, χωρίς σωστή διαχείριση ενδέχεται κάποια εικονικά μηχανήματα να απασχολούν όλους τους διαθέσιμους πόρους, αφήνοντας άλλα μηχανήματα σε σημείο να μην μπορούν να λειτουργήσουν. Η ζήτηση σε μνήμη αποτελεί ένα από τα κυριότερα σημεία ανταγωνισμού ανάμεσα στα εικονικά μηχανήματα ενός κόμβου. Έχουν καταστρωθεί κατά καιρούς αρκετοί τρόποι ώστε οι προγραμματιστές μνήμης (memory schedulers) στους επιβλέποντες των εικονικών μηχανημάτων να καταφέρνουν να προβλέπουν τις απαιτήσεις σε μνήμη των μηχανημάτων. Η πρόβλεψη των απαιτήσεων μνήμης οδηγεί τους προγραμματιστές μνήμης στη λήψη καλύτερων αποφάσεων σε ό,τι αφορά στο διαμοιρασμό των πόρων μνήμης ανάμεσά τους [7].

Συχνά, η ενεργά χρησιμοποιούμενη μνήμη αξιοποιείται ως εργαλείο πρόβλεψης των απαιτήσεων μνήμης. Ορίζουμε ως μέγεθος του συνόλου εργασίας ή ενεργά χρησιμοποιούμενη μνήμη ενός εικονικού μηχανήματος το σύνολο των σελίδων του εικονικού μηχανήματος στις οποίες πραγματοποιούνται συχνότερα προσβάσεις [9]. Προκειμένου να γίνεται κατά το δυνατόν αποδοτικότερη διαχείριση της μνήμης, είναι σημαντικό το μέγεθος του συνόλου εργασίας κάθε εικονικού μηχανήματος να προσεγγίζεται με ακρίβεια σε κάθε χρονική στιγμή.

Πλήθος από λύσεις έχουν προταθεί ανά καιρούς που αποσκοπούν στον υπολογισμό του μεγέθους του συνόλου εργασίας ενός εικονικού μηχανήματος. Μία πρόταση υλοποιημένη στον VMWare ESX hypervisor είναι η προσέγγιση του συνόλου εργασίας στατιστικά, με δειγματοληψία των προσβάσεων που συμβαίνουν σε σελίδες και εξαγωγή του ποσοστού προσβάσεων. Η ιδέα είναι ότι με αυτόν τον τρόπο γίνεται μια εκτίμηση των προσβάσεων που συμβαίνουν δεδομένης μιας αρχικής εκχώρησης μνήμης. Μια άλλη, περισσότερο υποσχόμενη προσέγγιση, είναι το χτίσιμο καμπυλών Miss Ratio [7].

2.2.1 Ορισμός MRC καμπυλών

Οι βοηθητικές καμπύλες που σχετίζονται με τη μνήμη αποθήκευσης (cache ή DRAM) συνιστούν σπουδαία εργαλεία για την διαχείριση των εκχωρήσεων μνήμης. Τέτοιου είδους καμπύλες αναπαριστούν κάποια μετρική απόδοσης ως συνάρτηση του μεγέθους της μνήμης.

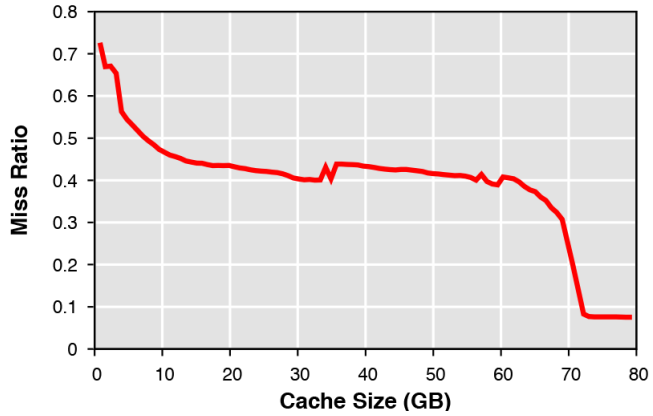
Οι MRC καμπύλες αφορούν είτε cache μνήμη, για την περίπτωση μελέτης διεργασιών, είτε κύρια μνήμη, για την περίπτωση μελέτης εικονικών μηχανημάτων. Οι MRC καμπύλες στοχευμένες σε μελέτη διεργασιών, αναπαριστούν την αναλογία των απωλειών (miss ratio) στη μνήμη cache ως προς τις ολικές αναφορές ενός φόρτου εργασίας, συναρτήσει του μεγέθους της μνήμης. Οι MRC καμπύλες στοχευμένες σε μελέτη εικονικών μηχανημάτων, αναπαριστούν το miss ratio δεδομένης της συνολικής μνήμης του εικονικού μηχανήματος ως προς τις ολικές αναφορές ενός φόρτου εργασίας, συναρτήσει του μεγέθους της μνήμης.

Από τον ορισμό τους, οι MRC καμπύλες είναι πολύ χρήσιμες στον προσδιορισμό του όγκου των δεδομένων που χρησιμοποιούνται σε ένα συγκεκριμένο φόρτο εργασίας, δηλαδή στον προσδιορισμό της ενεργά χρησιμοποιούμενης μνήμης[10].

Με χρήση των MRC καμπυλών, το πρόβλημα βελτιστοποίησης των εκχωρήσεων μνήμης σε περιβάλλοντα εικονικοποίησης, για ένα host κόμβο με N εικονικά μηχανήματα, ανάγεται στον υπολογισμό της μνήμης $\mathbf{m} = [m_1, m_2, ..., m_N]$ η οποία ελαχιστοποιεί τον αριθμό των απωλειών στη μνήμη:

Να ελαχιστοποιηθεί η τιμή

$$F(\mathbf{m}) = \sum_{i=1}^{N} mrc_i(m_i) * NR_i \tag{2.1}$$



Σχήμα 1: Παράδειγμα MRC καμπύλης για διεργασίες [11]

υπό τον περιορισμό

$$\sum_{i=1}^{N} m_i \leq M \tag{2.2}$$

για εφαρμογές, μέγιστη διαθέσιμη μνήμη M και $mrc_i(m_i)$ ο λόγος απώλειας σε δεδομένο μέγεθος μνήμης m_i. Αν NR_i ο συνολικός αριθμός των αιτημάτων για μια εφαρμογή, τότε ο αριθμός $mrc_i(m_i) * NR_i$ αντιστοιχεί στο συνολικό αριθμό απωλειών [12].

Οι Miss Ratio Καμπύλες, εκτός του ότι δύνανται να παρέχουν επακριβή αποτελέσματα στον υπολογισμό της ενεργά χρησιμοποιούμενης μνήμης, προσφέρουν και εκτίμηση των επιπτώσεων που θα έχουν στην απόδοση ποικιλόμορφες εκχωρήσεις μνήμης στο σύστημα ή στην εφαρμογή [13]. Συνεπώς, ο διαχειριστής του συστήματος μπορεί να αξιοποιήσει μια Miss Ratio καμπύλη που αφορά ολόκληρο το σύστημα ώστε να καθοριστεί το συνολικό μέγεθος της μνήμης με στόχο μια επιθυμητή βελτίωση στην καθολική απόδοση του συστήματος. Ομοίως, ένας αυτοματοποιημένος διαχειριστής μνήμης μπορεί να αξιοποιήσει ξεχωριστές Miss Ratio καμπύλες για ποικίλα φορτία εργασίας διαφορετικής προτεραιότητας, ώστε να βελτιστοποιήσει τις εκχωρήσεις μνήμης δυναμικά, αποσκοπώντας σε στόχους επιπέδου εφαρμογών (service-level objectives) [14], [11].

Δυστυχώς, το κόστος της εξαγωγής των MRC καμπυλών μπορεί να προκύψει πολύ μεγάλο [7], σε σημείο να αντισταθμίζει τα οφέλη που προκύπτουν από την μετέπειτα αξιοποίησή τους.

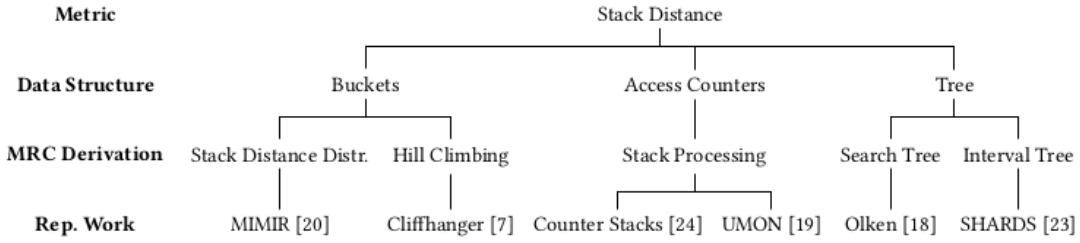
2.3 Αλγοριθμικοί μέθοδοι υπολογισμού MRC καμπυλών

Οι αλγοριθμικοί μέθοδοι υπολογισμού των MRC καμπυλών έχουν προσελκύσει σημαντικά το ενδιαφέρον της επιστημονικής κοινότητας. Πλήθος από αλγορίθμους έχουν αναπτυχθεί με στόχο την ελαχιστοποίηση της πολυπλοκότητας, προκειμένου να είναι εφικτό οι MRC καμπύλες να υπολογίζονται δυναμικά χωρίς να εισάγουν σε μεγάλο βαθμό επιπρόσθετες καθυστερήσεις. Οι αλγόριθμοι υπολογισμού των MRC καμπυλών ταξινομούνται σε δύο ευρύτερες κατηγορίες. Η πρώτη κατηγορία αφορά μεθόδους που χρησιμοποιούν ως μετρική την απόσταση στοίβας, ενώ η δεύτερη κατηγορία αφορά μεθόδους που βασίζονται στο χρόνο επαναχρησιμοποίησης. Οι ανωτέρω μετρικές ορίζονται ως εξής:

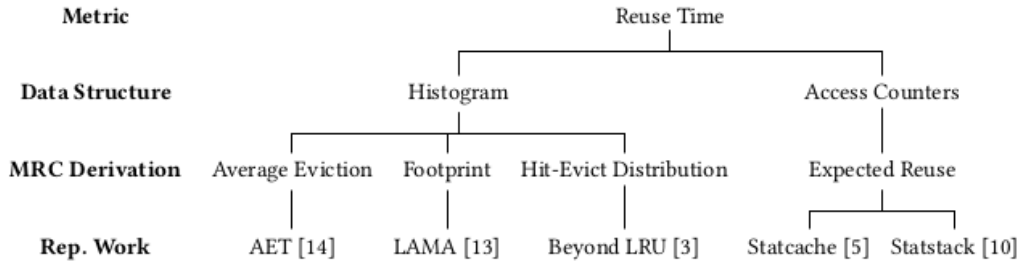
- **Απόσταση Στοίβας**: Το πλήθος των διαφορετικών προσβάσεων που παρεμβάλλονται ανάμεσα σε δύο προσβάσεις στο ίδιο αντικείμενο. Ονομάζεται εναλλακτικά και απόσταση επαναχρησιμοποίησης.

- **Χρόνος Επαναχρησιμοποίησης**: Ο ολικός αριθμός των προσβάσεων που παρεμβάλλονται ανάμεσα σε δύο προσβάσεις στο ίδιο αντικείμενο.

Στη συνέχεια παρατίθενται διαγραμματικά αλγόριθμοι που εμπίπτουν στις δύο κατηγορίες.



Σχήμα 2: Αλγόριθμοι βασισμένοι στην απόσταση στοίβας [10]



Σχήμα 3: Αλγόριθμοι βασισμένοι στον χρόνο επαναχρησιμοποίησης [10]

Η μετάθεση της προσοχής σε μεθόδους που βασίζονται στον χρόνο επαναχρησιμοποίησης αποτελεί μια σύγχρονη τάση, καθώς μπορεί να υπολογιστεί πιο εύκολα συγκριτικά με την απόσταση στοίβας, αλλά μέχρι πρόσφατα ([15]) ήταν σχεδόν ανέφικτη η εξαγωγή ακριβών αποτελεσμάτων δεδομένων των διαθέσιμων αναλυτικών μοντέλων.

Η ανάλυση των αποστάσεων επαναχρησιμοποίησης έχει αποδειχτεί ότι αποτελεί έναν εξαιρετικό μηχανισμό για την πρόβλεψη της συμπεριφοράς μνήμης των προγραμμάτων σε διαφορετικά σύνολα δεδομένων εισόδου [16], [17]. Η κατηγορία των αλγορίθμων που υπολογίζουν MRC καμπύλες βάσει της απόστασης επαναχρησιμοποίησης υλοποιούν πολιτική αντικαταστάσεων LRU ώστε να διατηρείται η σειρά της στοίβας. Προκειμένου να εξαχθεί η MRC καμπύλη δεδομένου ενός μεγέθους μνήμης, δημιουργείται μια κατανομή αποστάσεων στοίβας. Εφόσον υπάρχει γνώση της κατανομής, είναι εφικτός ο υπολογισμός των επιτυχημένων προσβάσεων σε μέγεθος μνήμης d ως το άθροισμα όλων των αναφορών με απόσταση στοίβας $\leq d$. Ομοίως, είναι εφικτός ο υπολογισμός των αποτυχημένων προσβάσεων ως το άθροισμα όλων των αναφορών με απόσταση στοίβας $\geq d$.

Αναλυτικότερα, αν d είναι το μέγεθος της μνήμης, N ο ολικός αριθμός των αιτημάτων και M το πλήθος των διαφορετικών αιτημάτων, τότε η αναλογία απώλειας στο μέγεθος d εκφράζεται ως:

$$(1 - \frac{\sum_{i=1}^{d} freq(i)}{N}) = \frac{\sum_{i=d}^{M} freq(i)}{N} \tag{2.3}$$

Η παρούσα εργασία εξετάζει την μέθοδο δέντρων που προτείνει ο Olken [18] με χρήση της απόστασης στοίβας ως μετρική. Επομένως, θα επεξηγηθεί το θεωρητικό υπόβαθρο με αφετηρία τον αλγόριθμο στοίβας του Mattson και έπειτα περιγραφή κάποιων αλγοριθμικών υλοποιήσεων με χρήση αναπαραστάσεων με δέντρα, συμπεριλαμβανομένου του αλγόριθμου του Olken, που οδηγούν στη βελτιστοποίησή του.

2.3.1 Πολιτικές στοίβας

Το 1970 οι Mattson κ.α. ασχολήθηκαν με μια σειρά από πολιτικές αντικατάστασης σελίδων, ονομαζόμενες ως πολιτικές στοίβας, για τις οποίες είναι εφικτή η αποτίμηση της συνάρτησης επιτυχιών για όλα τα μεγέθη της προσωρινής μνήμης σε ένα μοναδικό πέρασμα του ίχνους των αναφορών. Οι πολιτικές στοίβας διακρίνονται από μονότονα αυξανόμενο λόγο επιτυχιών συναρτήσει ενός αυξανόμενου μεγέθους προσωρινής μνήμης. Η πολιτική της στοίβας καθορίζει μια καθολική διάταξη σε όλες τις σελίδες που έχουν αναφερθεί μέχρι την παρούσα χρονική στιγμή. Η διάταξη αυτή ονομάζεται λίστα προτεραιότητας. Για ένα δεδομένο μέγεθος προσωρινής μνήμης, η πολιτική της στοίβας θα επιλέξει προς αντικατάσταση τη σελίδα εκείνη που διακρίνεται από την μικρότερη προτεραιότητα. Οι Mattson κ.α. απέδειξαν ότι αυτό ισοδυναμεί με την απαίτηση οι πολιτικές στοίβας να ικανοποιούν την εξής ιδιότητα: Για όλα τα ίχνη διευθύνσεων, ένα δοθέν μέγεθος προσωρινής μνήμης περιλαμβάνει όλες τις σελίδες που ανήκουν σε μικρότερα μεγέθη μνήμης σε ένα χρονικό σημείο. Ως αντιπαράδειγμα, την ιδιότητα αυτή δεν την ικανοποιεί η πολιτική FIFO, οπότε δεν εντάσσεται στις πολιτικές στοίβας. Αποτέλεσμα της ιδιότητας αποτελεί η ικανότητα αναπαράστασης των περιεχομένων σε όλα τα μεγέθη της προσωρινής μνήμης σε συγκεκριμένο χρονικό σημείο, με μια μοναδική στοίβα μεγέθους ίσου με το μέγεθος της μεγαλύτερης μνήμης [18].

2.3.2 Πολιτικές προτεραιότητας TIR

Οι πολιτικές προτεραιότητας TIR - Time Invariant Relative χαρακτηρίζονται από την ιδιότητα οι σχετικές προτεραιότητες των τμημάτων να είναι ανεξάρτητες του χρόνου. Πιο επεξηγηματικά, οι σχετικές προτεραιότητες ανάμεσα σε δύο τμήματα δεν διαφοροποιείται αν δεν προηγηθεί αναφορά σε κάποιο από αυτά. Κάθε πολιτική στην οποία η προτεραιότητα κάθε τμήματος προκύπτει ως συνάρτηση αποκλειστικά του ιστορικού έως την τελευταία αναφορά αυξανόμενου κατά κάποια σταθερά επί το χρόνο που παρήλθε από την τελευταία αναφορά στο ίδιο τμήμα ονομάζεται Time Invariant πολιτική. Επομένως, σε κάθε αναφορά η μόνη αλλαγή στη λίστα προτεραιότητας είναι η θέση του πιο πρόσφατα αναφερόμενου τμήματος. Κατά συνέπεια, για τέτοιου είδους πολιτικές είναι εφικτή η αναπαράσταση της λίστας προτεραιότητας ως σωρό (heap). Αν οι σχετικές προτεραιότητες δεν ήταν time invariant θα έπρεπε σε κάθε νέα αναφορά να γίνει ανοικοδόμηση του σωρού, αντί για απλή διαγραφή και επανένταξη ενός στοιχείου. Στα πλαίσια ενός σωρού, η εύρεση και η διαγραφή του στοιχείου με την εκάστοτε μικρότερη προτεραιότητα μπορεί να γίνει σε χρόνο $O(logC)$ όπου C ο αριθμός των τμημάτων στη μνήμη. Έτσι οδηγούμαστε σε συνολικό χρόνο υπολογισμού $O(nlogC)$ [18].

2.3.3 LRU πολιτική

Δεδομένου ότι η σχετική διάταξη δύο σελίδων στη στοίβα είναι time invariant, πηγάζει η συνειδητοποίηση ότι η διάταξη κατά σειρά προτεραιότητας και η διάταξη των στοιχείων της στοίβας θα είναι πανομοιότυπες. Έχει αποδειχθεί πως η ιδιότητα αυτή ικανοποιείται αποκλειστικά από την LRU πολιτική. Έστω ότι μια πολιτική που δημιουργεί στοίβα που είναι time invariant ως προς τις σχετικές σχέσεις των σελίδων αλλά δεν διατηρείται η σειρά προτεραιότητας. Τότε, είναι πιθανή η ύπαρξη δύο σελίδων που είναι εκτός της σειράς προτεραιότητας. Κάθε αναφορά σε σελίδα που βρίσκεται βαθύτερα στη στοίβα, θα οδηγούσε σε αναδιάταξή της (αφαίρεση όλων των στοιχείων και επανεισαγωγή ώστε να διατηρείται η σειρά προτεραιότητας), γεγονός που αντιπαρέρχεται την ιδιότητα του time invariance ως προς τις σχετικές θέσεις στοίβας [18].

2.3.4 Αλγόριθμος στοίβας του Mattson [19]

Ο αλγόριθμος στοίβας του Mattson είναι ο πρώτος αλγόριθμος που βασίζεται στην απόσταση στοίβας για τον υπολογισμό του λόγου της αποτυχίας για συγκεκριμένη χωρητικότητα της μνήμης προσωρινής αποθήκευσης. Ο αλγόριθμος αναπτύσσεται στα εξής πέντε βασικά βήματα, για κάθε αναφορά σε ένα δεδομένο [10]:

- **Βήμα 1** Αναζήτηση της στοίβας για εύρεση της παρούσας θέσης του δεδομένου (αν υπάρχει)
- **Βήμα 2** Υπολογισμός της απόστασης d από την κορυφή της στοίβας
- **Βήμα 3** Ανανέωση των μετρητών που αφορούν απώλειες σε όλα τα μεγέθη $\geq d$
- **Βήμα 4** Ανανέωση των μετρητών που αφορούν επιτυχίες σε όλα τα μεγέθη $\leq d$
- **Βήμα 5** Τοποθέτηση του δεδομένου στην κορυφή της στοίβας και αφαίρεσή του από την προηγούμενη θέση του (αν υπάρχει)

request	time	stack	stack dist.
a	1	a	∞
b	2	b,a	∞
c	3	c,b,a	∞
d	4	d,c,b,a	∞
a	5	a,d,c,b	3 (d,c,b)
a	6	a,d,c,b	0 (none)
d	7	d,a,c,b	1 (a)
b	8	b,d,a,c	3 (d,a,c)

Σχήμα 4: Παράδειγμα υπολογισμού της απόστασης επαναχρησιμοποίησης για ένα σύνολο αιτημάτων [10]

Ο αλγόριθμος εμφανίζει πολυπλοκότητα $O(NM)$, όπου N οι ολικές αναφορές και M το συνολικό πλήθος των διαφορετικών αναφορών. Πλήθος από εργασίες έχουν γίνει οι οποίες βελτιώνουν τον παραπάνω αλγόριθμο με αξιοποίηση πιο αποδοτικών δομών δεδομένων, δειγματοληπτική λήψη αιτημάτων και πρόβλεψη των αποστάσεων στοίβας [10]. Επίσης, έχει αποδειχτεί [20] ότι το φορτίο εργασίας ενός τυπικού προγράμματος μπορεί να προβλεφθεί επακριβώς δεδομένου ενός σημαντικά μικρότερου υποσυνόλου από αναφορές.

2.3.5 Αλγόριθμος Bennett και Kruskal [18], [21]

Η ελάχιστη χωρητικότητα μνήμης (MMC) ορίζεται ως το ελάχιστο μέγεθος της προσωρινής μνήμης που θα απαιτούνταν για να συμπεριληφθεί σε αυτήν μια σελίδα που εντοπίζεται στη στοίβα. Συνεπώς, ισοδυναμεί με το συνολικό βάθος έως την τοποθεσία της σελίδας στη στοίβα, αν μετρήσουμε από την κορυφή της.

Ο αλγόριθμος των Bennett και Kruskal υλοποιήθηκε το 1975 και στηρίχθηκε στην παρατήρηση ότι στην LRU πολιτική, οι ελάχιστες χωρητικότητες μνήμης ισούνται με τον αριθμό των μοναδικών σελίδων που έχουν αναφερθεί έως την τελευταία αναφορά στην υπό εξέταση σελίδα.

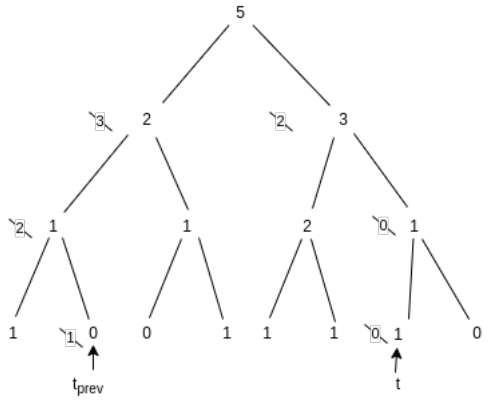
Οι Bennett και Kruskal κατασκεύασαν αρχικά ένα διάνυσμα bits μεγέθους όσο το ίχνος των προσβάσεων. Σε πρώτη φάση όλα τα bits είναι 0. Όταν κάποια πρόσβαση λάβει χώρα σε χρόνο t, το bit $b[t]$ γίνεται 1 και το bit $b[t_{prev}]$ που αναπαριστά την προηγούμενη αναφορά στην αναφερόμενη σελίδα εκκαθαρίζεται. Κατά συνέπεια, σε κάθε χρονικό σημείο όλα τα bits είναι 0 εκτός από αυτά που αντιστοιχούν στις πλέον πρόσφατες προσβάσεις κάθε σελίδας. Οι ελάχιστες χωρητικότητες μνήμης μπορούν έτσι να υπολογιστούν δεδομένης μιας αναφοράς με μέτρηση του πλήθους των άσσων που μεσολάβησαν από το παρόν χρονικό σημείο έως την τελευταία πρόσβαση στη σελίδα.

Ο αλγόριθμος των Bennett και Kruskal υπολογίζει με αποδοτικό τρόπο τις ελάχιστες χωρητικότητες μνήμης για μια αναφορά. Για να το επιτύχουν αυτό, κατασκεύασαν ένα δέντρο με φύλλα το διάνυσμα από bits . Το πλήθος των παιδιών του κάθε κόμβου επηρεάζει την εικόνα των μερικών αθροισμάτων αλλά δεν επηρεάζει την υπολογιστική διαδικασία, οπότε επαφίεται προς επιλογή. Η μοναδική ιδιότητα που πρέπει το δέντρο να διατηρεί είναι κάθε εσωτερικός του κόμβος να περιλαμβάνει το άθροισμα των άσσων που είναι αποθηκευμένα στα παιδιά τους. Κατά συνέπεια, αν επιλεχθούν διαστήματα μεγέθους m, το άθροισμα των m πρώτων στοιχείων του διανύσματος από bits διατηρείται

στον αριστερότερο κόμβο του δέντρου του αμέσως ανώτερου επιπέδου και ούτω καθ' εξής [21]. Η διαδικασία κατασκευής πρέπει να γίνει με τρόπο ώστε να μην υπάρχουν επικαλύψεις σε στοιχεία του διαστήματος m, οι οποίες θα οδηγούσαν σε διπλότυπους υπολογισμούς ίδιων στοιχείων στα μερικά αθροίσματα.

Ο υπολογισμός της θέσης της σελίδας στη στοίβα (ή αλλιώς η ελάχιστη χωρητικότητα μνήμης) γίνεται με αναγνώριση της προηγούμενης πρόσφατης αναφοράς t_{prev} στη συγκεκριμένη σελίδα (μέσω ευρετηρίου) και έπειτα διάσχιση από το φύλλο $b[t_{prev}]$ έως τον πλησιέστερο κοινό πρόγονο του δέντρου των φύλλων $b[t]$ και $b[t_{prev}]$. Στην πορεία της διάσχισης, για κάθε κόμβο που περιλαμβάνει το φύλλο $b[t_{prev}]$ στο αριστερό του υποδέντρο, προστίθεται το μέγεθος του δεξιού υποδέντρου στο τελικό αποτέλεσμα. Επίσης πριν την έναρξη της υπολογιστικής διαδικασίας πραγματοποιείται αλλαγή στους μετρητές $b[t]$ και $b[t_{prev}]$ και ανανεώνονται κατά πώς πρέπει οι ενδιάμεσοι κόμβοι που επηρεάζονται από την αλλαγή αυτή (αύξηση ή μείωση κατά 1). Σημειώνεται ότι για προσβάσεις που δεν αφορούν σε αρχική αναφορά σε μια σελίδα, η ανανέωση των μετρητών απαιτεί διάσχιση μόνο έως τον πλησιέστερο κοινό πρόγονο διότι από το σημείο αυτό και έπειτα, αλληλοαναιρούνται η μείωση κατά 1 bit λόγω του $b[t_{prev}]$ και η αύξηση κατά 1 bit λόγω του $b[t]$.

Στη συνέχεια παρατίθεται μια εικόνα στιγμιότυπου του δέντρου για $m = 2$ και προσβάσεις στην ίδια σελίδα σε χρόνους t_{prev} και t.



Σχήμα 5: Δέντρο μερικών αθροισμάτων - Αλγόριθμος Bennet-Kruskal

Ο συγκεκριμένος αλγόριθμος, όπως περιγράφηκε από τους Bennett και Kruskal δεν υποστηρίζει πεπερασμένα μεγέθη προσωρινής μνήμης και συγχρόνως οσοδήποτε μεγάλο ίχνος διευθύνσεων. Δεδομένης της υπόθεσης ότι ενδιαφερόμαστε αποκλειστικά για μεγέθη μνήμης μικρότερα από C σελίδες, ανά C αναφορές θα πρέπει να γίνει συμπίεση του διανύσματος bits ώστε να είναι συνεχόμενο και ανοικοδόμηση του δέντρου ώστε να ξεκινάει με το νωρίτερο t_{prev}. Η διαδικασία συμπίεσης εισάγει πολυπλοκότητα $O(C)$ αλλά πραγματοποιείται ανά C αναφορές, οπότε η συνολική πολυπλοκότητα που εισάγει είναι μόνο $O(n)$. Πλέον το ύψος του δέντρου είναι φραγμένο σε ύψος $O(logC)$, οπότε λαμβάνουμε τελική πολυπλοκότητα χείριστης περίπτωσης $O(nlogC)$.

2.3.6 Αλγόριθμος Αναζήτησης Δέντρων του Olken [18]

Υπάρχει η δυνατότητα αποδοτικής αναπαράστασης της στοίβας ως δυαδικό δέντρο όταν η διάταξη των σελίδων στη στοίβα είναι time invariant. Στην περίπτωση κατά την οποία η διάταξη των σελίδων στοίβας δεν ήταν time invariant, τότε έπειτα από κάποια αναφορά σε σελίδα θα απαιτούταν ανοικοδόμηση του δέντρου έναντι απλής διαγραφής και επανεισαγωγής της αναφερόμενης σελίδας [18].

Ακριβώς αυτή είναι η αλγοριθμική ιδέα του Olken, και αφορά στην αναπαράσταση της LRU στοίβας ως ένα δυαδικό δέντρο βασισμένο στον χρόνο προσβάσεων, με τρόπο τέτοιον ώστε μια inorder διάσχιση του δέντρου να δίνει τη θέση των δεδομένων στην LRU στοίβα. Η inorder διάσχιση ορίζεται ως η διάσχιση όλων των κόμβων του δέντρου με αναδρομικό τρόπο, διασχίζοντας πρώτα το

αριστερό υποδέντρο, έπειτα τον κόμβο ρίζα και τέλος το δεξί υποδέντρο. Κάθε κόμβος που ανήκει στο δέντρο αναπαριστά μια σελίδα και το δέντρο είναι ταξινομημένο βάσει της τελευταίας αναφοράς σε κάθε σελίδα. Έτσι, δεδομένης μιας αναφοράς στην ρίζα του δέντρου, το δεξί υποδέντρο αναπαριστά αναφορές που έγιναν πιο πρόσφατα ενώ το αριστερό υποδέντρο αναπαριστά παλαιότερες αναφορές.

Κάθε φορά που πραγματοποιείται αναφορά σε μια σελίδα, αρχικά γίνεται εντοπισμός του αντίστοιχου κόμβου στο δέντρο με χρήση ευρετηρίου, καθορισμός της θέσης του στη στοίβα όπως αναλύεται στη συνέχεια, διαγραφή και επανένταξη στο δέντρο στην κορυφή της στοίβας - στο δεξιότερο παιδί. Το δέντρο μπορεί να προκύψει εξαιρετικά μη ισορροπημένο, οπότε επιλέγεται η χρήση AVL δέντρου, αλλά και διαφορετικά είδη δέντρων μπορούν επίσης να αξιοποιηθούν. Το AVL δέντρο εγγυάται ότι το ύψος του δέντρου είναι λογαριθμικό ($O(logM)$ αν M ο αριθμός των κόμβων του). Επιπλέον, η πολυπλοκότητα χείριστης περίπτωσης για εισαγωγή και διαγραφή οποιουδήποτε κόμβου είναι $O(logM)$. Συνεπώς, ο υπολογισμός της θέσης της σελίδας στη στοίβα, η διαγραφή της και η επανένταξη στο δέντρο μπορούν να ολοκληρωθούν εντός χρόνου $O(logM)$.

Προς διευκόλυνση του καθορισμού της θέσης της σελίδας εντός της στοίβας, γίνεται αποθήκευση σε κάθε κόμβο του δέντρου του αριθμού των σελίδων που κείτονται στο υποδέντρο του κόμβου. Το δέντρο είναι ταξινομημένο βάσει του χρόνου της τελευταίας αναφοράς, οπότε όλες οι σελίδες του δεξιότερου υποδέντρου μιας σελίδας έχουν αναφερθεί πιο πρόσφατα από αυτήν. Συνεπώς, το σύνολο των σελίδων που έχουν αναφερθεί πιο πρόσφατα δεδομένης μιας σελίδας, έστω p, εμπεριέχουν την p στο αριστερό τους υποδέντρο, ή βρίσκονται στο δεξιότερο υποδέντρο της σελίδας που εμπεριέχει το p στο αριστερό της υποδέντρο. Έτσι, η θέση της σελίδας p στη στοίβα μπορεί να ανακτηθεί με διάσχιση του μονοπατιού από την αναφερόμενη σελίδα έως τη ρίζα του δέντρου. Η χρήση δεικτών προς τον πατέρα κάθε κόμβου θα μπορούσε να αποφευχθεί με εναλλακτική αναζήτηση στο δέντρο από τη ρίζα προς τη σελίδα p με αξιοποίηση του τελευταίου χρόνου αναφοράς -ανακτώμενου από το ευρετήριο- ως κλειδί. Στην πορεία της διάσχισης, εύρεση κόμβου που εμπεριέχει το p στο αριστερό υποδέντρο (έχει αναφερθεί πιο πρόσφατα από τη σελίδα p για παράδειγμα) γίνεται άθροιση στο τελικό αποτέλεσμα του μεγέθους του αριστερού υποδέντρου του κόμβου συν 1 -για τον ίδιο τον κόμβο. Τέλος, στο αποτέλεσμα προστίθεται και το μέγεθος του δεξιότερου υποδέντρου του κόμβου της σελίδας p. Το παραπάνω άθροισμα δίνει τη θέση της σελίδας p τη στοίβα.

Έτσι, ο αλγόριθμος δίνει χρονική πολυπλοκότητα $O(NlogM)$ και καταλαμβάνει χώρο $O(M)$, όπου N οι ολικές αναφορές και M το συνολικό πλήθος των διαφορετικών αναφορών και συνεπώς αποτελεί βελτίωση στον αλγόριθμο του Mattson χωρίς να θυσιάζει στην ακρίβεια.

2.4 Συστήματα υπολογισμού MRC καμπυλών

Στο παρόν τμήμα γίνεται συνοπτική παρουσίαση κάποιων υπαρχόντων συστημάτων και τεχνικών που επικεντρώνονται στην κατασκευή MRC καμπυλών.

2.4.1 Το σύστημα Geiger

Το σύστημα Geiger [22] αναπτύσσει τεχνικές που μπορούν να αξιοποιηθούν από τον διαχειριστή των εικονικών μηχανημάτων (hypervisor ή VMM) ώστε να καταλήξει σε χρήσιμες πληροφορίες που αφορούν στην buffer cache και το σύστημα εικονικής μνήμης (virtual memory system) του φιλοξενούμενου εικονικού μηχανήματος. Όσο η εικονικοποίηση εφαρμόζεται ολοένα και περισσότερο, ο hypervisor φυσιολογικά υποκαθιστά το λειτουργικό σύστημα, ως βασικός διαχειριστής των πόρων του εικονικού μηχανήματος. Στην ουσία, το σύστημα Geiger προσπαθεί να δώσει λύση στην έλλειψη πληροφορίας εκ μέρους του VMM για τα λειτουργικά συστήματα των εικονικών μηχανημάτων, κατά το χρόνο εκτέλεσης. Συγκεκριμένα, περιγράφονται τεχνικές που οδηγούν σε ικανότητα παρατήρησης του VMM των αλληλεπιδράσεων του φιλοξενούμενου λειτουργικού συστήματος με οντότητες όπως η MMU και οι συσκευές αποθήκευσης. Λόγω του ότι ο VMM εικονικοποιεί δραστηριότητες I/O προς τον δίσκο, τα γεγονότα γραψίματος και διαβάσματος στον δίσκο που συμβαίνουν στο φιλοξενούμενο εικονικό σύστημα είναι εμφανή στον VMM. Επίσης, είναι εμφανείς και οι ενημερώσεις του πίνακα

σελίδων. Χάρη στη γνώση αυτή, οι τεχνικές του Geiger καταλήγουν σε συμπεράσματα σχετικά με το πότε εισάγονται και εξάγονται σελίδες από την buffer cache του φιλοξενούμενου συστήματος. Οι τεχνικές αναγνώρισης των evictions από την buffer cache των μηχανημάτων, είναι χρήσιμες και για την υλοποίηση του MemRx, που είναι ένας VMM ο οποίος αναγνωρίζει WSS των φιλοξενούμενων μηχανημάτων. Συγκεκριμένα, το MemRx χρησιμοποιεί το Geiger για ανίχνευση των evictions αλλά και των διαδοχικών επαναφορτώσεων από την buffer cache του φιλοξενούμενου μηχανήματος. Έτσι, το MemRx ποσοτικοποιεί τον αριθμό των προσβάσεων μνήμης που μπορούν να μετατραπούν από αστοχίες σε ευστοχίες για διαφορετικά μεγέθη μνήμης, χτίζοντας την MRC καμπύλη με διατήρηση ουράς για τα evictions που παρατηρούνται. Οι αποστάσεις επαναχρησιμοποίησης που υπολογίζονται κατά τη διαδικασία χτισίματος της MRC καμπύλης, είναι προσεγγιστικά όση ο πραγματικός αριθμός των evictions που συνέβησαν ανάμεσα στο eviction της σελίδας και της επαναφόρτωσής της. Το σύστημα Geiger, όμως, δεν έχει εικόνα σε περιπτώσεις όπου το μέγεθος του συνόλου εργασίας είναι μικρότερο από το αρχικό μέγεθος σε μνήμη του εικονικού μηχανήματος [3].

2.4.2 Το σύστημα SHARDS

Μέχρι στιγμής, η ελάχιστη πολυπλοκότητα των αλγοριθμικών τεχνικών που υπολογίζουν καμπύλες MRC είναι ασυμπτωτικά $O(NlogM)$ ως προς το χρόνο και $O(M)$ ως προς το χώρο, όπου N το πλήθος των συνολικών αναφορών και M το πλήθος των μοναδικών αναφορών. Το σύστημα SHARDS [14], που λαμβάνει το όνομά του από τα αρχικά της φράσης Spatially Hashed Approximate Reuse Distance Sampling μειώνει σημαντικά τις χωρικές και τις χρονικές απαιτήσεις του αλγόριθμου εξαγωγής MRC καμπυλών καθιστώντας την ενσωμάτωση της συνεχούς, δυναμικής δημιουργίας της καμπύλης στο λογισμικό των συστημάτων εφικτή. Επιπλέον, επιτρέπει την ανάλυση μεγάλου όγκου προσβάσεων που εξαιτίας περιορισμών σε μνήμη ήταν ανεφάρμοστη στο παρελθόν.

Το σύστημα SHARDS εισήγαγε μια νέα προσέγγιση στην ανάλυση της απόστασης επαναχρησιμοποίησης που στηρίζεται σε τυχαία χωρική δειγματοληψία σελίδων, παρεχόμενης με ανίχνευση προσβάσεων σε αντιπροσωπευτικές τοποθεσίες. Η επιλογή των τοποθεσιών που θα ληφθούν υπόψιν στον τελικό υπολογισμό πραγματοποιείται δυναμικά, βάσει συνάρτησης επί της τιμής κατακερματισμού τους. Η προσεγγιστική κατασκευή της MRC καμπύλης με την τεχνική SHARDS επιτρέπει σε σύνολα αναφορών τάξης πολλών GB να δίνουν ακριβείς καμπύλες με ανίχνευση συνόλου μεγέθους μικρότερου από 1MB.

Η βασική ιδέα πάνω στην οποία χτίστηκε το σύστημα SHARDS είναι εννοιολογικά απλή: Για κάθε αναφορά σε τοποθεσία L, η απόφαση αν θα συμπεριληφθεί η τοποθεσία L στο δειγματοληπτικό σύνολο λαμβάνεται με βάση αν η τιμή $hash(L)$ ικανοποιεί μια συνθήκη. Έτσι, για παράδειγμα, η συνθήκη $hash(L) mod 100 < K$ οδηγεί σε δειγματοληψία του $K\%$ του συνολικού πλήθους των τοποθεσιών.

Η συγκεκριμένη μέθοδος εξασφαλίζει την ανίχνευση όλων των προσβάσεων που αναφέρονται στην ίδια τοποθεσία, μιας και όλες οι προσβάσεις έχουν την ίδια τιμή κατακερματισμού. Επιπλέον, δεν απαιτείται καμία πληροφορία τόσο ως προς το σύνολο των τοποθεσιών που ενδέχεται να αναφερθούν δεδομένου του φορτίου εργασίας, όσο και ως προς την κατανομή των προσβάσεων στις επιλεγείσες τοποθεσίες. Σαν αποτέλεσμα, η δειγματοληψία SHARDS είναι ανεπηρέαστη από την εκάστοτε κατάσταση.

Πιο αναλυτικά ως προς τη συνθήκη υπολογισμού, αν αυτή είναι της μορφής $hash(L) mod P < K$ με χρήση του υπολοίπου P και για κατώφλι T ο λόγος δειγματοληψίας προκύπτει ως $R = T/P$ και κάθε δείγμα αντιπροσωπεύει στατιστικά $1/R$ των συνολικών τοποθεσιών. Μια σημαντική ιδιότητα που πηγάζει από τη δομή της συγκεκριμένης συνθήκης είναι ότι με ελάττωση του κατωφλιού σε τιμή $T' < T$, τα επιλεγέντα δείγματα στο κατώφλι T' ανήκουν σε υποσύνολο των επιλεγέντων δειγμάτων στο κατώφλι T. Επίσης, προκειμένου η απόσταση επαναχρησιμοποίησης να υπολογίζεται σωστά, θα πρέπει να κλιμακωθεί κατάλληλα με τον παράγοντα $1/R$ εφόσον κάθε τοποθεσία- δείγμα αναπαριστά στατιστικά ένα μεγαλύτερο αριθμό τοποθεσιών. Καταληκτικά, δεδομένου ότι N οι ολικές αναφορές και M οι μοναδικές αναφορές, με τη μέθοδο SHARDS αν ορίσουμε έναν σταθερό ρυθμό δειγματοληψίας R το προβλεπόμενο πλήθος των μοναδικών τοποθεσιών που επιλέχτηκαν προς δειγματοληψία

θα ισούται με $R \times M$. Δεδομένου ότι το λαμβανόμενο δείγμα είναι επαρκώς αντιπροσωπευτικό, ο ολικός αριθμός των αναφορών στις σελίδες - δείγματα μειώνεται αναλογικά σε περίπου $R \times N$. Για τα περισσότερα φορτία εργασίας η τεχνική SHARDS φανερώνει ότι τιμή του $R = 0.001$ δίνει MRC καμπύλη μεγάλης ακρίβειας με χρήση μνήμης και πόρων επεξεργασίας που είναι τάξεις μεγέθους μικρότερες συγκριτικά με συμβατικές προσεγγίσεις.

Επίσης, στο [14] παρουσιάζεται και η περαιτέρω επέκταση της μεθόδου SHARDS με καθορισμό ενός ανώτατου ορίου στο μέγεθος της απαιτούμενης μνήμης. Βασική ιδέα είναι η προσαρμοστική ελαχιστοποίηση του ρυθμού δειγματοληψίας με στόχο τη διατήρηση ενός σταθερού ορίου s_{max} στο συνολικό αριθμό των τοποθεσιών - δειγμάτων που ανιχνεύονται. Η υλοποίηση απαιτεί παρακολούθηση του συνόλου S των τοποθεσιών με διατήρηση δυάδων της μορφής $\langle L_i, T_i \rangle$, όπου $T_i = hash(L_i) mod P$ και L_i η τοποθεσία - δείγμα. Ο ρυθμός δειγματοληψίας αρχικοποιείται σε μια σχετικά μεγάλη τιμή R, η οποία στη συνέχεια θα ελαττώνεται δυναμικά. Για κάθε τοποθεσία που ανιχνεύεται για πρώτη φορά, εξετάζεται αν το μέγεθος του συνόλου S υπερβαίνει το φράγμα s_{max} . Σε αυτή την περίπτωση, γίνεται αφαίρεση της δυάδας $\langle L_j, T_{max} \rangle$ που αντιστοιχεί στην ανώτερη τιμή για το κατώφλι T από το διατηρούμενο σύνολο και ακολούθως το κατώφλι μειώνεται στη νέα τιμή T_{max}, επιβάλλοντας έτσι και νέα τιμή στο ρυθμό δειγματοληψίας $R_{new} = T_{max}/P$. Με χρήση του προκαθορισμένου μεγέθους των δειγμάτων, η χρονική και χωρική πολυπλοκότητα είναι εφικτό να μειωθούν αντιστοίχως έως τις τιμές $O(M)$ και $O(1)$.

Τελικά, η μέθοδος SHARDS είναι μια μέθοδος ακριβής, εύρωστη και οδηγεί σε σημαντική βελτίωση της απόδοσης. Η ανάλυση της απόδοσης όπως παρουσιάζεται στο [14] αποκαλύπτει δραματική ελάττωση στην κατανάλωση πόρων: έως $10.800\times$ σε μνήμη και έως $204\times$ σε CPU.

2.4.3 Δυναμική απόφαση ενεργοποίησης μηχανισμού λήψης μετρικών

Ο W.Zhao κ.α. [13] μελέτησαν την κατασκευή MRC καμπυλών βασιζόμενη σε LRU πολιτική και με χρήση AVL δέντρων για εικονικά μηχανήματα. Η μέθοδός τους υλοποιήθηκε στο επίπεδο του Xen επιβλέποντα. Όταν ένα παρά-εικονικοποιημένο εικονικό μηχάνημα προσπαθήσει να ανανεώσει τον πίνακα σελίδων του, εκτελεί hypercall αίτημα ώστε να ειδοποιήσει τον hypervisor να εφαρμόσει τη ζητούμενη ενημέρωση. Κατά τη μέθοδο που προτείνουν, όταν ένα τέτοιο request ανιχνευθεί στον hypervisor, αρχικά γίνεται η ενημέρωση και στη συνέχεια γίνεται ανάκληση της άδειας πρόσβασης (access permission) μέσω του bit στον πίνακα σελίδων. Έπειτα, όταν το λειτουργικό σύστημα του εικονικού μηχανήματος επιχειρήσει πρόσβαση στη σελίδα, οδηγούμαστε σε minor pagefault που ανιχνεύεται αρχικά από τον hypervisor. Έτσι χτίζεται η MRC καμπύλη και η άδεια πρόσβασης αποκαθίσταται. Προκειμένου να ελαττωθεί περαιτέρω η καθυστέρηση που εισάγει η κατασκευή της MRC καμπύλης, βασίστηκαν στη γνώση ότι πλήθος προγραμμάτων εμφανίζουν σημαντική χωρική και χρονική τοπικότητα, γεγονός που έχει ως αποτέλεσμα το μέγεθος του συνόλου εργασίας να είναι σχετικά σταθερό για μεγάλα διαστήματα. Στηρίχτηκαν, έτσι, στην εμφάνιση φάσεων εντός προγραμμάτων [9] με παρόμοια ζήτηση σε πόρους μνήμης. Η ιδέα είναι ότι σε περιόδους κατά τις οποίες το πρόγραμμα διανύει μια φάση, μπορεί προσωρινά να απενεργοποιείται ο μηχανισμός λήψης μετρικών για την κατασκευή της MRC καμπύλης χωρίς να επηρεαστεί η ακρίβειά της. Κλειδί στην υλοποίηση αποτέλεσε η πρόβλεψη των αλλαγών φάσης στα προγράμματα, ώστε να σηματοδοτούν την επανενεργοποίηση του μηχανισμού. Εν τέλει, αξιοποίησαν γεγονότα σχετιζόμενα με το υλικό (απώλειες στο TLB , στην L1 και στην L2 μνήμη) ώστε να εκτιμούν την αλλαγή φάσεων. Οδηγήθηκαν, έτσι, σε πολύ καλή ακρίβεια κατασκευής της MRC καμπύλης και με πολύ μικρές καθυστερήσεις.

2.4.4 MRC καμπύλες με αξιοποίηση bits προσβάσεων

Αρκετές έρευνες έχουν ασχοληθεί με την κατασκευή MRC καμπυλών με χρήση των bits πρόσβασης (accessed bits), υλοποιημένες είτε στο λειτουργικό σύστημα [23] είτε στον επιβλέποντα τον εικονικών μηχανημάτων [7].

Επέκταση του VMKernel

Ο Reth κ.α. [7] έχουν εντάξει αλλαγές στον πυρήνα του VMware ESX επιβλέποντα, τον VMKernel. Στο σύστημα αυτό χρησιμοποιείται επίσης η μέθοδος SHARDS, για καθορισμό ενός δείγματος σελίδων στις οποίες θα υπολογιστούν οι αποστάσεις επαναχρησιμοποίησης. Ο καθορισμός των δειγμάτων γίνεται βάσει του Physical Page Number - PPN σελίδων κατά το πρώτο pagefault σε αυτές. Μέσα στον μηχανισμό διαχείρισης λαθών του VMKernel, η σελίδα πάνω στην οποία έγινε το pagefault μπορεί να ανιχνευτεί και να καταγραφεί.Υπάρχει η δυνατότητα αφαίρεσης των αντιστοιχίσεων μεταξύ των εικονικών διευθύνσεων και τον PPN ανά σελίδα τόσο στον πίνακα σελίδων όσο και στο TLB, ώστε να πυροδοτούνται συνεχώς pagefaults με κάθε πρόσβαση, που θα μας οδηγήσει πρακτικά σε ικανότητα ανίχνευσης όλων των προσβάσεων. Αυτή η μέθοδος, όμως, θα αύξανε σημαντικά τους χρόνους εκτέλεσης των διεργασιών στο εικονικό μηχάνημα, αφού θα έπρεπε μονίμως να γίνεται ανανέωση των εισόδων του πίνακα σελίδων και του TLB σύμφωνα με τα δεδομένα στο δίσκο. Ως εναλλακτική προσέγγιση, πραγματοποιείται περιοδικά διάσχιση των accessed bits στον πίνακα σελίδων, για το προκαθορισμένο δείγμα σελίδων. Σε επεξεργαστές που υποστηρίζουν accessed bits, το accessed bit για μια σελίδα τίθεται αυτόματα όταν γίνεται πρόσβαση σε αυτήν. Η σελίδα στην οποία εντοπίστηκε πρόσβαση καταγράφεται για τον υπολογισμό της MRC καμπύλης κατά την περιοδική διαδικασία, και το accessed bit μηδενίζεται για ανίχνευση επόμενων προσβάσεων. Έπειτα, οι είσοδοι του TLB εκκαθαρίζονται ώστε επόμενη πρόσβαση σε σελίδα να επιφέρει αστοχία σε αυτό. Αυτό θα οδηγήσει τον μηχανισμό σελιδοποίησης του VMKernel να συμβουλευτεί τον πίνακα σελίδων για να θέσει την TLB είσοδο, και επομένως θα τεθεί ξανά το accessed bit, που θα διαβαστεί στην επόμενο περιοδικό πέρασμα και ούτω καθ' εξής. Κατά συνέπεια, υπάρχει αύξηση στον αριθμό των TLB αστοχιών, όταν η περιοδική διαδικασία διάσχισης των accessed bits συμβαίνει συχνά. Το overhead εμφανίζει αλλαγές με αλλαγή στο πλήθος των δειγμάτων, αλλά και με αλλαγή της συχνότητας σύμφωνα με την οποία συμβαίνει η περιοδική διαδικασία.

Δυναμική απόκτηση MRC καμπυλών για διεργασίες

Κινούμενοι σε παρόμοια λογική, οι P. Zhou κ.α. [23] πρότειναν δύο συστήματα για τον υπολογισμό MRC καμπυλών για διεργασίες, με τον αλγόριθμο του Mattson [19] με δυναμικό τρόπο. Πρόκειται για την πρώτη μέθοδο υπολογισμού της MRC καμπύλης σε χρόνο εκτέλεσης στην εικονική μνήμη. Επίσης, πρόκειται για την πρώτη μέθοδο που αξιοποιεί την καμπύλη ώστε να δρομολογήσει την εκχώρηση μνήμης και να διαχειριστεί την ενέργεια μνήμης αποσκοπώντας στην ελαχιστοποίηση της ενεργειακής κατανάλωσης. Σε πρώτη φάση, υλοποίησαν έναν μηχανισμό επίβλεψης σε επίπεδο υλικού, με εξειδικευμένες δομές προσωρινής αποθήκευσης που ενημερώνουν τα περιεχόμενά τους σε επιτυχημένες ή αποτυχημένες προσβάσεις σελίδων. Σε δεύτερη φάση, εισήγαγαν αλλαγές στο σύστημα της εικονικής μνήμης του Linux για την παροχή λύσης αποκλειστικά σε επίπεδο λογισμικού. Ομοίως με το [7], στηρίχτηκαν επίσης στην ιδέα της περιοδικής εκκαθάρισης του accessed bit για ανίχνευση των προσβάσεων σε σελίδες. Το δείγμα σελίδων για τις οποίες θα ελέγχεται το accessed bit δίνεται με διαφορετικό τρόπο συγκριτικά με SHARDS. Πιο συγκεκριμένα, διατηρούν μια scan list δομή με σελίδες, για τις οποίες θα γίνει διάσχιση του accessed bit. Προκειμένου να ανιχνεύσει προσβάσεις σε σελίδες που δεν ανήκουν στην scan list, το λειτουργικό σύστημα σβήνει την read-write άδεια για αυτές τις σελίδες. Έτσι, προσβάσεις σε αυτές οδηγούν σε σφάλμα άδειας (protection fault), οπότε η σελίδα ανιχνεύεται και τοποθετείται στην scan λίστα. Επίσης, η read-write άδεια επανατίθεται ώστε να μην οδηγηθούμε σε επόμενα σφάλματα άδειας. Η τελευταία σελίδα της scan list εξάγεται από αυτήν, με ταυτόχρονο σβήσιμο της read-write άδειάς της. Επίσης, στο σύστημα αυτό επέκτειναν περαιτέρω τις προσπάθειες ελαχιστοποίησης της εισαγόμενης καθυστέρησης, ορίζοντας ως βασική μονάδα εξεταζόμενης μνήμης ομάδες σελίδων αντί για απλές σελίδες, προσέγγιση που ακολούθησαν και στην λύση σε επίπεδο υλικού.

Κεφάλαιο 3

Το σύστημα FluidMem

Στο παρόν κεφάλαιο πραγματοποιείται λεπτομερής παρουσίαση του συστήματος FluidMem ([3], [24], [4]) το οποίο επεκτείνει η παρούσα διπλωματική. Ειδικότερα, περιγράφεται η συνεισφορά του συστήματος και αναλύονται τα δομικά του μέρη.

3.1 Βασική ιδέα του FluidMem

Τα προηγούμενα χρόνια παρατηρήθηκε αύξηση του ενδιαφέροντος [25], [26] γύρω από την επίλυση του προβλήματος παροχής ολοένα και περισσότερης μνήμης σε εφαρμογές που εμφανίζουν υψηλή ζήτηση, σε σημείο που υπερβαίνει τους πόρους που μπορεί να διαθέσει ένας υπολογιστικός κόμβος. Ως αποτέλεσμα, ξεκίνησαν σταδιακά προσπάθειες ενσωμάτωσης απομακρυσμένης μνήμης, δηλαδή μνήμης που βρίσκεται σε άλλους υπολογιστικούς κόμβους, με στόχο την αύξηση της χωρητικότητας.

Κάπως έτσι, γεννήθηκε η ιδέα του Memory Dissaggregation, ως ένα μοντέλο στο οποίο οι υπολογιστικές μονάδες μπορούν να συντεθούν από διακριτές ποσότητες μνήμης προερχόμενες από πολλούς διαφορετικούς υπολογιστικούς κόμβους. Πλήθος από προσπάθειες έχουν γίνει με στόχο την πραγμάτωση αυτής της ιδέας. Εναλλακτικές σχεδιαστικές επιλογές σε επίπεδο υλικού έχουν μελετηθεί ώστε να παρέχουν αυτό το μοντέλο, αλλά απαιτούν την ύπαρξη νέων υποδομών στα υπολογιστικά κέντρα. Τα τελευταία χρόνια, όμως, οι εξελίξεις στις τεχνολογίες δικτύων έχουν οδηγήσει στην μετάθεση της προσοχής σε προσεγγίσεις λογισμικού.

Πάνω σε αυτή την ιδέα δημιουργήθηκε το FluidMem, ως ένα σύστημα ανοιχτού πηγαίου κώδικα που φτιάχτηκε με στόχο να αποδείξει ότι η λογική του ολικού διαχωρισμού της μνήμης ενός φυσικού μηχανήματος από τις εφαρμογές που τρέχουν σε αυτό είναι εφικτή και υλοποιήσιμη, ενώ επέκτεινε την ιδέα σε περιβάλλοντα εικονικοποίησης καθιστώντας ανεξάρτητη την μνήμη των φιλοξενούμενων εικονικών μηχανημάτων από την φυσική μνήμη που προσφέρει ο υπολογιστικός κόμβος στον οποίο ανήκουν. Η μετάθεση του προβλήματος σε περιβάλλοντα εικονικοποίησης προέκυψε από την συνειδητοποίηση ότι το στρώμα εικονικοποίησης παρέχει μια βολική αφαίρεση για να εισαχθεί η έννοια της απομακρυσμένης μνήμης και να αποσυνδεθεί η εικονική μνήμη από την φυσική μνήμη.

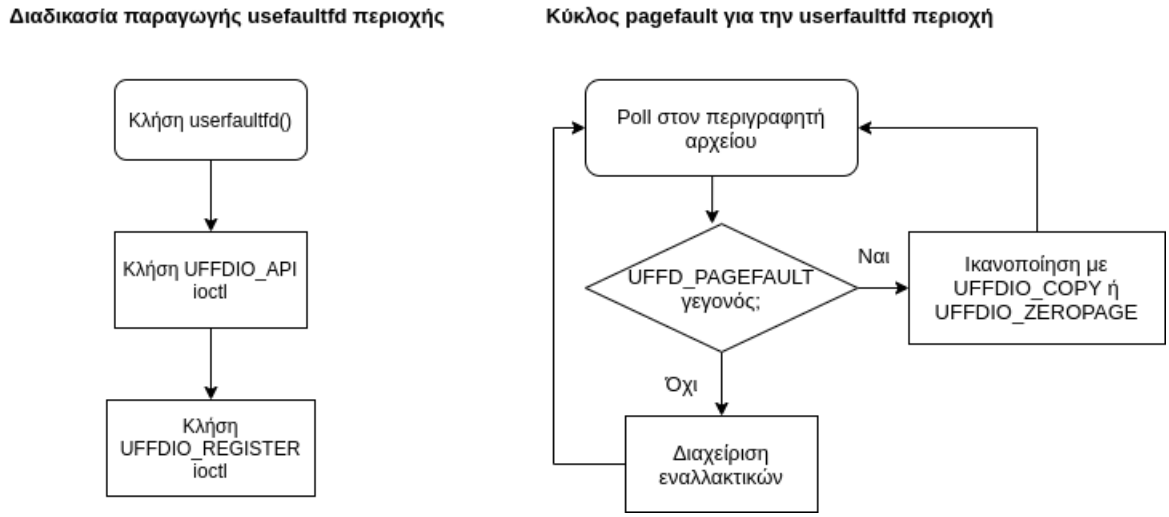
Στα υπόλοιπα τμήματα του κεφαλαίου παρουσιάζονται τα δομικά στοιχεία του FluidMem, ως ένα σύστημα που παρέχει Full Memory Dissaggregation σε επίπεδο λογισμικού και που απαιτεί μηδενικές αλλαγές σε επίπεδο υλικού. Πιο συγκεκριμένα, θα αναλυθούν τα προϋπάρχοντα συστήματα και οι μηχανισμοί που το FluidMem επέλεξε να χρησιμοποιήσει καθώς και οι λόγοι που έγιναν οι συγκεκριμένες επιλογές. Παράλληλα, θα περιγραφούν οι διεπαφές που παρέχονται από το FluidMem με στόχο την επικοινωνία της κεντρικής monitor διεργασίας του FluidMem με τα επιμέρους συστήματα.

3.2 Σελιδοποίηση σε επίπεδο χρήστη

Η σελιδοποίηση σε επίπεδο χρήστη είναι και η σχεδιαστική απόφαση που συνετέλεσε καθοριστικά στην παροχή ολικού memory disaggregation. Το FluidMem αξιοποιεί μια σχετικά νέα δυνατότητα του πυρήνα του Linux, γνωστή ως userfaultfd [27], προκειμένου να αναγνωρίζει και να ικανοποιεί λάθη σελίδων από εφαρμογές που ανήκουν σε επίπεδο χρήστη. Ως εναλλακτική επιλογή, συστήματα όπως το Infiniswap [2] επιλέγουν να αξιοποιήσουν τη διεπαφή swap του Linux, προκειμένου να εντάξουν την αφαίρεση της απομακρυσμένης μνήμης. Η επιλογή αυτή, όμως, εισάγει έναν εγγενή περιορισμό, μιας και δύναται να παρέχει μόνο μερικό διαχωρισμό της μνήμης.

Υπάρχουν δύο βασικές διαφορές ανάμεσα στον ολικό και τον μερικό διαχωρισμό της μνήμης. Αρχικά, σε ένα μοντέλο πλήρους διαχωρισμού μνήμης όλες οι σελίδες της μνήμης είναι ικανές να διαμοιραστούν σε κάποιον άλλο υπολογιστικό κόμβο. Η σελιδοποίηση σε απομακρυσμένη μνήμη μέσω της διεπαφής swap έρχεται σε αντιδιαστολή με αυτή την ιδιότητα δεδομένου ότι ο διαμοιρασμός σε άλλους κόμβους είναι επιτρεπτός αποκλειστικά και μόνο σε ανώνυμες σελίδες. Συνεπώς, πολλά άλλα είδη σελίδων όπως σελίδες αρχείων ή σελίδες που ανήκουν στον πυρήνα του λειτουργικού συστήματος δεν είναι δυνατό να διαμοιραστούν σε άλλους κόμβους. Το FluidMem είναι ικανό να υποστηρίξει πλήρη διαχωρισμό της μνήμης για οποιαδήποτε σελίδα που ανήκει σε ένα εικονικό μηχάνημα. Έπειτα, σε ένα μοντέλο πλήρους διαχωρισμού μνήμης δεν θα πρέπει να απαιτείται η συνεργασία του φιλοξενούμενου εικονικού μηχανήματος για αποφάσεις που αφορούν στη διαχείριση της μνήμης. Αυτό έχει ως αποτέλεσμα ένας πάροχος του Cloud να μπορεί να διαχειρίζεται τη μνήμη πολλών εικονικών μηχανημάτων, χωρίς την ανάγκη αλλαγών στο φιλοξενούμενο λειτουργικό σύστημα ή συνεργασίας με τις διάφορες εφαρμογές που τρέχουν σε αυτό. Συμπερασματικά, το μοντέλο πλήρους διαχωρισμού μνήμης επιτρέπει σε έναν πάροχο τόσο να αυξάνει όσο και να μειώνει τη διανομή της μνήμης, καθιστώντας το αποτύπωμα της μνήμης ενός εικονικού μηχανήματος ικανό να κλιμακωθεί σε πολλούς υπολογιστικούς κόμβους.

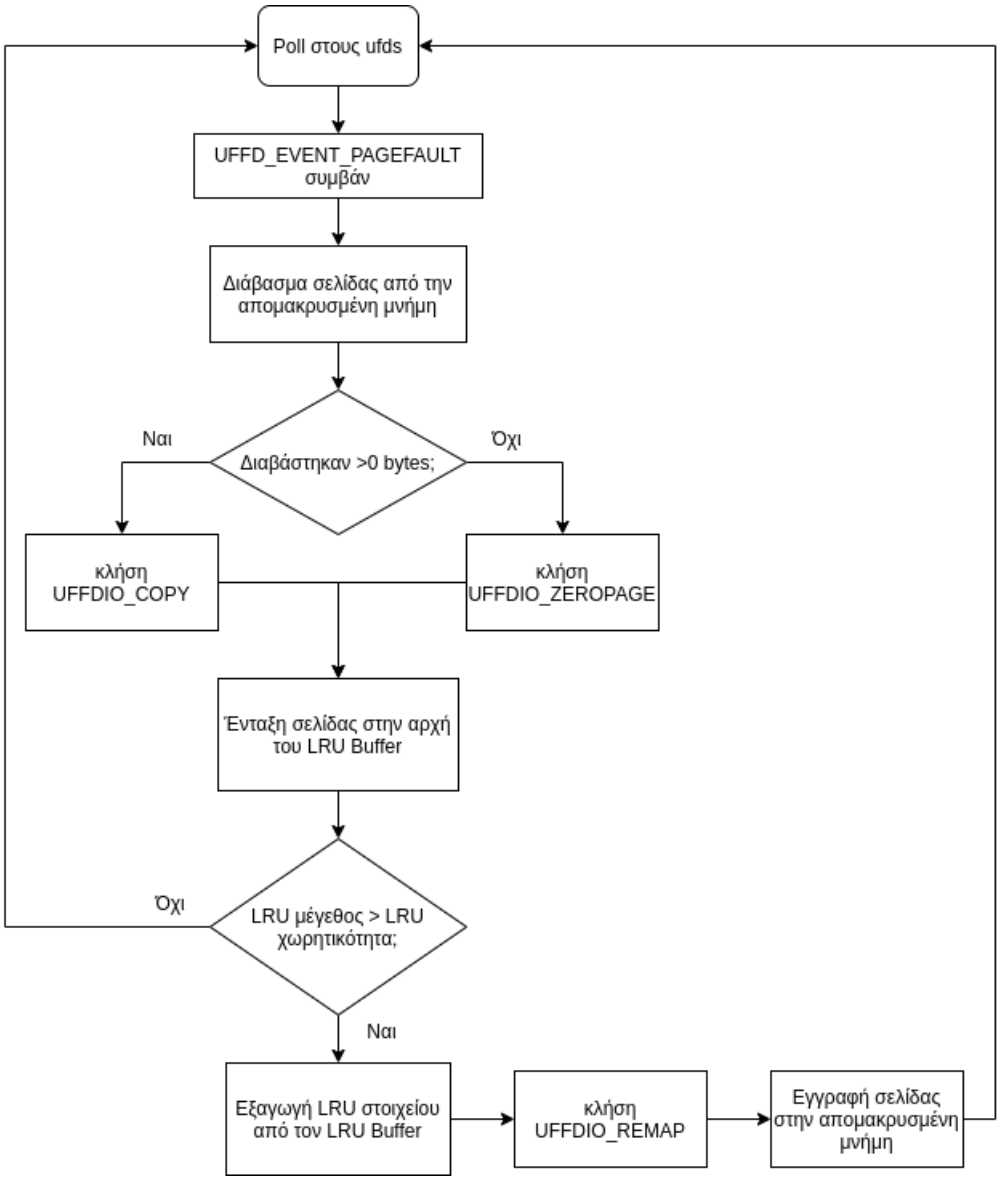
Ο μηχανισμός userfaultfd ενσωματώθηκε στον πυρήνα του Linux από την έκδοση 4.3 και έπειτα. Ο μηχανισμός καθιστά εφικτή τη σελιδοποίηση σε επίπεδο χρήστη αξιοποιώντας περιγραφείς αρχείων (file descriptors) για να αναπαραστήσουν περιοχές διευθύνσεων της μνήμης. Ειδικότερα, ο μηχανισμός userfaultfd δύναται να επιβλέπει μια περιοχή διευθύνσεων μνήμης που έχει εκχωρηθεί σε μια εφαρμογή έπειτα από απευθείας αίτημα. Το αίτημα αποστέλλεται από την εφαρμογή για μια συγκεκριμένη περιοχή διευθύνσεων και αφορά την εγγραφή της περιοχής αυτής στον μηχανισμό. Πιο τεχνικά, σε πρώτη φάση γίνεται userfaultfd κλήση ώστε να αρχικοποιηθεί ένας νέος userfaultfd περιγραφητής αρχείου. Έπειτα, πραγματοποιείται UFFDIO_API ioctl κλήση ώστε να οριστεί σωστά η διεπαφή (παράδειγμα, αρχιτεκτονική 32 ή 64 bits) και να τεθούν ρητά κάποια χαρακτηριστικά που ενδέχεται να είναι από προεπιλογή απενεργοποιημένα στον πυρήνα. Το αίτημα εγγραφής, στη συνέχεια, γίνεται με UFFDIO_REGISTER ioctl κλήση και αφορά πλέον μια συγκεκριμένη περιοχή διευθύνσεων. Αποτέλεσμα ενός επιτυχημένου αιτήματος εγγραφής είναι ένας περιγραφητής αρχείου τον οποίο η εφαρμογή ή και κάποια άλλη εφαρμογή -επιπέδου χρήστη- μπορεί να χειριστεί κατά πώς επιθυμεί. Όταν ένα λάθος σελίδας λάβει χώρα μέσα στη συγκεκριμένη περιοχή μνήμης, το γεγονός αυτό θα επιστραφεί ως μήνυμα στον περιγραφητή αρχείου. Το μήνυμα θα παρέχει πληροφορίες για το υπό εξέταση λάθος, όπως η διεύθυνση της σελίδας στην οποία έγινε η πρόσβαση που οδήγησε σε αυτό αλλά και μια μάσκα από bits που τίθεται για παροχή γνώσης αν το λάθος αφορούσε γράψιμο της σελίδας.



Σχήμα 6: Διάγραμμα ροής για διαχείριση pagefaults μέσω του userfaultfd

Στο FluidMem, ο μηχανισμός userfaultfd αξιοποιείται με στόχο οποιοδήποτε λάθος σελίδας λαμβάνει χώρα, να ικανοποιείται με ανάλογο αίτημα στην απομακρυσμένη μνήμη. Έτσι, λάθη που αφορούν διάβασμα μιας σελίδας ικανοποιούνται από το μηχανισμό σελιδοποίησης του FluidMem με ανάκτησή της από την απομακρυσμένη μνήμη και τοποθέτηση στην τοπική μνήμη. Λάθη που αφορούν γράψιμο μιας σελίδας ικανοποιούνται πάντα με επιστροφή μιας κενής σελίδας στην εφαρμογή, ώστε να αποφευχθεί ο χρόνος πρόσβασης στην απομακρυσμένη μνήμη. Όταν μια σελίδα επιστρέφεται από την απομακρυσμένη μνήμη, λαμβάνεται επίσης και η απόφαση αν χρειάζεται να αφαιρεθεί μια σελίδα για να υπάρξει χώρος για τη νεοεισαχθείσα σελίδα. Η απόφαση στηρίζεται σε LRU πολιτική και η εφαρμογή της βασίζεται στη χρήση ενός LRU Buffer με μέγεθος που προκαθορίζεται, αλλά μπορεί να αλλαχθεί και δυναμικά έπειτα από αίτηση. Ο LRU Buffer αποθηκεύει εντός του τις εικονικές διευθύνσεις όλων των σελίδων που βρίσκονται στην τοπική μνήμη του κόμβου που τρέχει το FluidMem. Ο LRU Buffer είναι μοναδικός για όλα τα εικονικά μηχανήματα ή τις εφαρμογές που διαχειρίζεται το FluidMem. Έτσι, σελίδες από πλήθος εικονικών μηχανημάτων ή εφαρμογών βρίσκονται στον LRU Buffer κάθε χρονική στιγμή.

Προκειμένου να υποστηριχθεί η μεταφορά της σελίδας στην απομακρυσμένη μνήμη, ήταν απαραίτητο να ενσωματωθούν στον userfaultfd μηχανισμό κάποιες αλλαγές. Ειδικότερα, προστέθηκε η δυνατότητα μετακίνησης της περιοχής μνήμης που ελέγχει ο userfaultfd μηχανισμός σε διαφορετικές διευθύνσεις. Οι αλλαγές ενσωματώθηκαν με χρήση των remap patches του Andrea Arcangeli [28] και εισάγουν τη δυνατότητα κλήσης της UFFDIO_REMAP ioctl. Η συγκεκριμένη συνάρτηση ανανεώνει τον πίνακα σελίδων με remapping σελίδων που ανήκουν σε μια ανώνυμη εικονική περιοχή μνήμης (anonymous vma) σε κάποια άλλη. Στην περίπτωση του remapping, υπάρχουν δύο δυνατά σενάρια λειτουργίας που καθορίζονται με βάση το αν η UFFDIO_REMAP_MODE_DIRECTION_IN σημαία έχει τεθεί ή όχι. Για την περίπτωση της 'IN' κατεύθυνσης, η UFFDIO_COPY προτιμάται έναντι της UFFDIO_REMAP για την αντιγραφή μιας περιοχής vma χρήστη σε ένα εύρος userfaultfd περιοχής. Αυτό συμβαίνει διότι με την UFFDIO_REMAP κλήση απαιτείται εκκαθάριση του TLB για την περιοχή διευθύνσεων από την οποία θα γίνει το remapping, διαδικασία αποδεδειγμένα περισσότερο χρονοβόρα από την απλή αντιγραφή της. Η UFFDIO_REMAP κλήση είναι χρήσιμη στη συμπληρωματική περίπτωση της αφαίρεσης μνήμης από την userfaultfd περιοχή και την επιστροφή της σε μια περιοχή χρήστη. Επίσης, με θέσιμο της UFFDIO_REGISTER_MODE_MISSING σημαίας, διαδοχικές προσβάσεις στην ίδια περιοχή θα οδηγήσουν σε pagefaults. Το γεγονός αυτό επιτρέπει μη-συνεργατική αφαίρεση (eviction) μνήμης πάνω σε μια vma εγγεγραμμένη στον userfaultfd μηχανισμό, δίνοντας δυνατότητα για μείωση του ποσού της εκχωρημένης μνήμης στην περιοχή αυτή [28].



Σχήμα 7: Διάγραμμα ροής του μηχανισμού σελιδοποίησης του FluidMem

Το FluidMem παρέχει την βιβλιοθήκη libuserfault η οποία συμπεριλαμβάνει το αρχείο userfault-client.c. Οποιαδήποτε εφαρμογή θελήσει να στείλει αίτημα προς την monitor διεργασία εγγραφής στο FluidMem σύστημα, αρκεί να κάνει χρήση αυτής της βιβλιοθήκης επιπέδου χρήστη. Η monitor θα λάβει απλά τον περιγραφητή αρχείου όπως θα γραφεί σε ειδικό socket από τη userfault-client, και θα επιβλέπει και θα ικανοποιεί συμβάντα σε αυτόν.

3.3 Το σύστημα RAMCloud

Το FluidMem δύναται να υποστηρίξει περισσότερα από ένα συστήματα για το ρόλο της απομακρυσμένης μνήμης. Συγκεκριμένα, παρέχει τη βιβλιοθήκη libexternram που συμπεριφέρεται ως wrapper για τα διαφορετικά συστήματα. Το FluidMem έχει δοκιμαστεί με δύο διαφορετικά συστήματα, το memcached και το RAMCloud, εκ των οποίων επιλέγεται να αναλυθεί το ένα από αυτά -το σύστημα RAMCloud ([29], [30])- που δίνει και τα καλύτερα αποτελέσματα με Infiniband δίκτυο για το FluidMem [3] από άποψη απόδοσης. Το RAMCloud είναι ένα σύστημα αποθήκευσης βασισμένο στην DRAM το οποίο χτίστηκε με στόχο να παρέχει το ελάχιστο δυνατό χρόνο αποθήκευσης σε εφαρμογές κέντρου δεδομένων. Είναι γρήγορο και κλιμακώσιμο αρκετά ώστε να είναι σε θέση να

αξιοποιήσει τα μελλοντικά συστήματα δικτύων παρέχοντας εξαιρετικά χαμηλού χρόνου προσβάσεις σε πολύ μεγάλα σύνολα δεδομένων.

3.3.1 Συσκευές αποθήκευσης

Προκειμένου να υπάρχει η δυνατότητα αποθήκευσης συνόλου δεδομένων που υπερβαίνουν κατά πολύ τη χωρητικότητα ενός απλού μηχανήματος, το RAMCloud σχεδιάστηκε έτσι ώστε να συγκεντρώνει τη μνήμη DRAM εκατοντάδων χιλιάδων μηχανημάτων εντός ενός datacenter. Πληθώρα αποφάσεων πάρθηκαν στον τελικό σχεδιασμό του RAMCloud, που αποσκοπούν ακριβώς στον τελικό στόχο επίτευξης εξαιρετικά μικρού χρόνου αποθήκευσης, σε σημείο να είναι 100 - 1.000 φορές μικρότερο συγκριτικά με προηγούμενα συστήματα αποθήκευσης. Μια από τις καθοριστικές αποφάσεις είναι ότι όλα τα δεδομένα βρίσκονται στην κύρια μνήμη DRAM συνεχώς. Η επιλογή αυτή έγινε προκειμένου να ωθηθεί ο χρόνος πρόσβασης στα όριά του κατά το δυνατόν πιο επιθετικά, αξιοποιώντας την ίδια στιγμή τεχνολογίες solid state storage (SSS). Ταυτόχρονα, η ανθεκτικότητα του συστήματος επιτυγχάνεται με την διατήρηση αντιγράφων που αποθηκεύονται σε δίσκους άλλων συμμετεχόντων μηχανημάτων και τα οποία χρησιμοποιούνται αποκλειστικά και μόνο σε περιπτώσεις αποτυχίας κάποιου μηχανήματος.

3.3.2 Μνήμη αποτελούμενη από logs

Μια καθοριστική απόφαση στην επίτευξη των στόχων του RAMCloud ήταν η δόμηση της μνήμης με append-only logs. Εναλλακτικοί μηχανισμοί εκχώρησης μνήμης όπως η malloc και οι copying συλλέκτες σκουπιδιών κρίθηκαν ακατάλληλοι διότι είτε δεν επιτρέπουν την ανακατανομή αντικειμένων σε άλλα σημεία της μνήμης από τη στιγμή που έχουν αποθηκευτεί (εμφάνιση προβλημάτων κατακερματισμού μνήμης), είτε καταλήγουν κάποια στιγμή στο σημείο να πρέπει να διατρέξουν όλα τα εν ζωή δεδομένα, να τα ανακατανείμουν και να ενημερώσουν τις αναφορές τους.

Με το log-structured μοντέλο μνήμης, τα αντικείμενα που αποθηκεύονται στη μνήμη του RAMCloud είναι αμετάβλητα, γεγονός το οποίο υποδηλώνει ότι κάθε πράξη ανανέωσης της τιμής ενός δεδομένου ή διαγραφής αυτού σηματοδοτείται με μια νέα εισαγωγή στο τέλος των προηγούμενων logs. Όπως είναι αναμενόμενο, η διαχείριση των logs απαιτεί καθοριστικές αποφάσεις ως προς το ξεκαθάρισμα της μνήμης ώστε να είναι συνεχώς σε θέση το σύστημα να δεχτεί νέες εισαγωγές.

Σε πρώτη φάση, το σύστημα θα πρέπει να είναι ικανό να αντιγράφει αντικείμενα σε νέες θέσεις προκειμένου να διαχειρίζεται αποδοτικά τη μνήμη κάτω από οποιαδήποτε μοτίβα προσβάσεων ελαχιστοποιώντας τον κατακερματισμό. Δεύτερον, θα πρέπει να μην απαιτεί ένα καθολικό πέρασμα της μνήμης και αντί αυτού, να εκτελεί την αντιγραφή αυξητικά, σε μικρές περιοχές της μνήμης ανεξάρτητα, με κόστος ποσοστιαίο προς το μέγεθος της περιοχής. Ανάμεσα στα πλεονεκτήματά της, η αυξητική προσέγγιση επιτρέπει στο συλλέκτη σκουπιδιών να επικεντρωθεί σε περιοχές που έχουν τον περισσότερο ελεύθερο χώρο, γεγονός που βελτιώνει την αποδοτικότητά του. Τρίτον, η λειτουργία του ξεκαθαρίσματος θα πρέπει να μην καταστέλλει τη λειτουργία των νημάτων της εφαρμογής, ιδιαίτερα εκείνων που δεν σχετίζονται με την εκχώρηση νέας μνήμης. Σε κάθε περίπτωση, θα ήταν άνευ νοήματος το χτίσιμο ενός τόσο χαμηλού latency συστήματος όπως το RAMCloud με χρήση ενός μηχανισμού που προσθέτει σημαντικές διακυμάνσεις στο χρόνο απόκρισης. Το log-structured μοντέλο μνήμης του RAMCloud δρομολογεί όλες τις προαναφερθείσες απαιτήσεις. Παραδείγματος χάριν, η χρήση ενός εκκαθαριστή logs για την ανασυγκρότηση της ελεύθερης μνήμης συνιστά λύση σε προβλήματα κατακερματισμού. Αυτό σημαίνει ότι το RAMCloud είναι ικανό να προσαρμοστεί σε μεταβαλλόμενα μοτίβα προσβάσεων χωρίς να πέσει θύμα των ίδιων ανεπαρκειών σε μνήμη από τις οποίες υποφέρουν οι non-copying εκχωρητές. Ως αποτέλεσμα, η μνήμη μπορεί να χρησιμοποιηθεί αποδοτικά, χωρίς την ανάγκη να δοθεί εκ των προτέρων περισσότερη μνήμη σαν λύση των παθογενειών.

Επίσης, η log-structured προσέγγιση παρέχει αυξητική συλλογή σκουπιδιών, γεγονός που επιτρέπει στο RAMCloud να εκχωρεί μνήμη αποδοτικά ακόμη και υπό συνθήκες υψηλής χρήση μνήμης. Κατά συνέπεια, είναι απαραίτητο οι δείκτες σε ένα αντικείμενο να μπορούν να εντοπιστούν χωρίς να απαιτείται πέρασμα όλης της μνήμης. Ευτυχώς, τα συστήματα αποθήκευσης γενικά διαθέτουν αυτή

την ιδιότητα, κατά την οποία οι δείκτες περιορίζονται σε ευρετήρια όπου μπορούν με ευκολία να εντοπιστούν. Πιο συγκεκριμένα, στην περίπτωση του RAMCloud, κάθε κόμβος του οποίου η μνήμη συνεισφέρεται στην καθολική μνήμη του συστήματος διατηρεί έναν hash πίνακα όπου εντοπίζονται οι μοναδικοί απευθείας δείκτες στα αποθηκευμένα αντικείμενα. Επίσης, για κάθε αποθηκευμένο αντικείμενο υπάρχει ένας και μοναδικός δείκτης στον hash πίνακα. Είναι σημαντικό να σημειωθεί ότι ενώ οι παραδοσιακοί εκχωρητές μνήμης πρέπει να λειτουργούν και σε περιβάλλοντα στα οποία ο εκχωρητής δεν έχει κανέναν έλεγχο πάνω στους δείκτες, ένα log-structured σύστημα δεν θα μπορούσε να λειτουργήσει σε τέτοια περιβάλλοντα.

Είναι ακριβώς η περιορισμένη χρήση των δεικτών στο RAMCloud που καθιστά δυνατή την ταυτόχρονη συλλογή σκουπιδιών αποτρέποντας ολοκληρωτικά την καταστολή της εφαρμογής. Αναλυτικότερα, μόλις μια αναφορά στον hash πίνακα είναι αναγκαίο να τροποποιηθεί κατά τη μετακίνηση ενός αντικειμένου από τον εκκαθαριστή. Επιπλέον, τα δεδομένα είναι αμετάβλητα και αλλαγές σε αυτά αναπαρίστανται με τη λογική της αντιγραφής κατά το γράψιμο, οπότε μια απλή ατομική ανταλλαγή στον hash δείκτη του αντικειμένου είναι υπεραρκετή προκειμένου να πραγματοποιηθεί σύγχρονα ανακατάταξη ή ενημέρωσή του. Έτσι, δεν υπάρχει λόγος ανησυχίας για ανταγωνισμό ανάμεσα σε έναν εκκαθαριστή που μετακινεί ένα αντικείμενο και σε ένα αίτημα εγγραφής που το ανανεώνει.

Το μεγαλύτερο μέρος της καθυστέρησης που εμφανίζεται στην log-structured προσέγγιση σχετίζεται με τη συλλογή σκουπιδιών. Η εκχώρηση νέας μνήμης προς αποθήκευση είναι απλή υπόθεση, δεδομένου ότι τα νέα αντικείμενα τοποθετούνται απλά στο τέλος των προηγούμενων εγγραφών. Αντιθέτως, η αποδέσμευση του ελεύθερου χώρου είναι πιο απαιτητική από άποψη απόδοσης, διότι ο εκκαθαριστής πρέπει να αντιγράψει όλα τα ενεργά αντικείμενα που εντοπίζει στην περιοχή της μνήμης προς εκκαθάριση σε νέο σημείο για λόγους κατακερματισμού. Δυστυχώς, αυτό επιφέρει γρήγορη αύξηση του κόστους εκκαθάρισης όσο η χρήση μνήμης πλησιάζει στο μέγιστο. Το RAMCloud έχει εντάξει τεχνικές για καλή επιλογή των περιοχών μνήμης πάνω στις οποίες θα πραγματοποιηθεί η συλλογή σκουπιδιών, αλλά σε κάθε περίπτωση οποιοσδήποτε copying εκχωρητής υποφέρει από πολύ μεγάλες καθυστερήσεις όσο η χρήση της μνήμης πλησιάζει το μέγιστο σημείο.

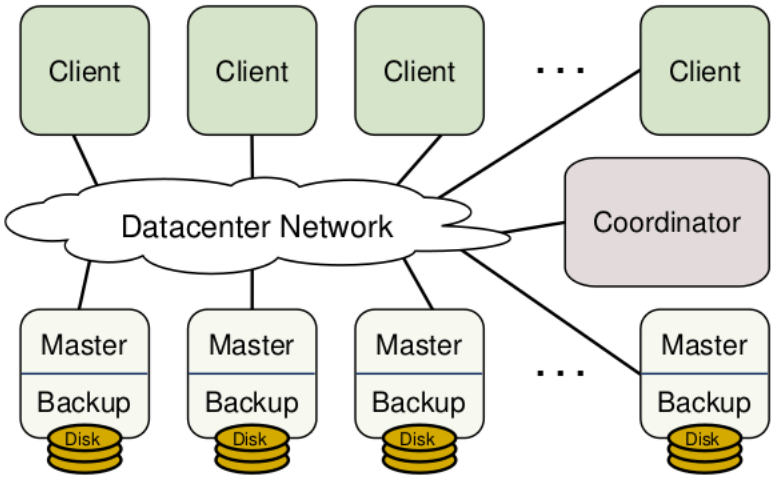
3.3.3 Εκκαθάριση δύο επιπέδων

Μια ακόμη βασική συνεισφορά του RAMCloud είναι η υλοποίηση της εκκαθάρισης σε δύο επίπεδα, δηλαδή η μνήμη και ο δίσκος να καθαρίζονται αυτόνομα. Αυτό σημαίνει ότι η μνήμη μπορεί να εκκαθαρίζεται χωρίς να απεικονίζονται οι αλλαγές στα αντίγραφα του δίσκου. Έτσι, η μνήμη μπορεί να έχει υψηλότερη χρήση από τον δίσκο. Η εκκαθάριση του δίσκου, αντιθέτως, συμβαίνει λιγότερο συχνά, έχοντας ως αποτέλεσμα οι εγγραφές στον δίσκο να είναι περισσότερες από αυτές της μνήμης, οπότε θα είναι και μικρότερη η συνολική μνήμη που χρησιμοποιείται. Η εκκαθάριση δύο επιπέδων συνδυάζει τα δυνατά σημεία της μνήμης και του δίσκου ως δομές αποθήκευσης, λαμβάνοντας συγχρόνων υπόψιν τις αδυναμίες τους. Όσον αφορά στη μνήμη, ο χώρος είναι πολύτιμος αλλά η καθυστέρηση για την εκκαθάριση δεν είναι μεγάλη, οπότε η εκκαθάριση συμβαίνει συχνά ώστε να επιτραπεί υψηλότερη χρήση. Όσον αφορά στο δίσκο, υπάρχει πολύς διαθέσιμος χώρος αλλά η καθυστέρηση της εκκαθάρισης είναι μεγαλύτερη, οπότε γίνεται χρήση του επιπλέον χώρου με στόχο την ελαχιστοποίησή της.

3.3.4 Δομικά μέρη και τοπολογία του RAMCloud

Τα δομικά μέρη του RAMCloud ταξινομούνται σε τρείς γενικές κατηγορίες. Οι οντότητες αυτές συνεργάζονται προκειμένου να παρέχουν ένα ανθεκτικό και συνεπές σύστημα αποθήκευσης για χρήση από τους πελάτες. Το RAMCloud απαρτίζεται από ένα σύνολο κόμβων αποθήκευσης, προτιθεμένων να διαθέσουν μέρος της μνήμης τους στην ολική μνήμη του συστήματος. Κάθε κόμβος αποθήκευσης συνίσταται από δύο ξεχωριστά δομικά μέρη. Το πρώτο δομικό μέρος είναι μια διεργασία master, υπεύθυνη για την διαχείριση της κύριας μνήμης του κόμβου και για την αποθήκευση σε αυτήν των αντικειμένων του RAMCloud. Η master διεργασία λαμβάνει και ικανοποιεί αιτήματα από τους πελάτες, όπως για διάβασμα ή για γράψιμο αντικειμένων. Το δεύτερο δομικό μέρος είναι μια

διεργασία backup, η οποία αξιοποιεί τον τοπικό δίσκο ή τη μνήμη flash για αποθήκευση αντιγράφων των δεδομένων που ανήκουν σε masters άλλων κόμβων. Τα δεδομένα αυτά δεν χρησιμοποιούνται κατά τη διάρκεια της φυσιολογικής λειτουργίας του συστήματος. Αντιθέτως, προσπελάζονται μόνο σε περιπτώσεις αναβίωσης ενός αποτυχημένου master κόμβου ή κατά την επανέναρξη ενός δικτύου που είχε καταστείλει την λειτουργία του.



Σχήμα 8: Τοπολογία του RAMCloud [29]

Η διαχείριση τόσο των masters όσο και των backups γίνεται από έναν κεντρικό συντονιστή (coordinator διεργασία) που πρωτίστως αναλαμβάνει θέματα configuration, όπως ποιοι είναι οι εγγεγραμμένοι στο δίκτυο κάθε στιγμή (masters και backups) αλλά και πώς κατανέμονται τα δεδομένα μεταξύ των masters. Άλλη βασική λειτουργία του συντονιστή είναι ο προσδιορισμός των masters και των backups που έχουν αποτύχει και η ενορχήστρωση της αναβίωσής τους. Σημειώνεται ότι ο συντονιστής δεν εμπλέκεται φυσιολογικά σε συνήθεις λειτουργίες όπως το διάβασμα και το γράψιμο δεδομένων. Οι χρήστες του RAMCloud διατηρούν τοπικά αντίγραφα της πληροφορίας του συντονιστή για την τοποθέτηση των δεδομένων προκειμένου να επικοινωνούν απευθείας με τους masters. Έτσι, απευθύνονται στον συντονιστή μόνο όταν τα τοπικά τους αντίγραφα δεν περιλαμβάνουν την απαραίτητη πληροφορία ή όταν τα αντίγραφα που διαθέτουν δεν είναι ενημερωμένα. Αυτό έχει ως αποτέλεσμα μείωση του φόρτου εργασιών του συντονιστή, κάτι που του επιτρέπει να κλιμακώνεται σε μεγάλα μεγέθη RAMCloud cluster. Επίσης, μειώνεται και ο χρόνος απόκρισης στους χρήστες, δεδομένου ότι στέλνουν απευθείας αίτημα στον master που διατηρεί το επιθυμητό μέρος των δεδομένων. Το RAMCloud υποστηρίζει επίσης μια ξεχωριστή υπηρεσία για παροχή πρωτοκόλλου συναίνεσης (consensus) σε κατανεμημένα συστήματα. Η υπηρεσία αυτή είναι υπεύθυνη για τη διαχείριση του συντονιστή και της κατάστασής του. Ειδικότερα, προκειμένου να αποφευχθεί αποτυχία όλου του συστήματος λόγω μιας αποτυχίας του συντονιστή, ο συντονιστής λειτουργεί πάνω από υψηλής ανεκτικότητας σε λάθη κατανεμημένο πρωτόκολλο, που συνίσταται από 3 έως 7 ξεχωριστούς κόμβους. Έτσι, ανά πάσα στιγμή, το πολύ ένας από αυτούς τους κόμβους ενστερνίζεται το ρόλο του συντονιστή, και οι υπόλοιποι κόμβοι είναι έτοιμοι να πάρουν τη θέση του σε περίπτωση ανάγκης. Ο καθορισμός του νέου συντονιστή αλλά και η διατήρηση συνεπών αντιγράφων της κατάστασης του τωρινού συντονιστή παρέχεται από την προαναφερθείσα ξεχωριστή υπηρεσία. Το RAMCloud είναι σχεδιασμένο να υποστηρίζει διάφορες υλοποιήσεις πρωτοκόλλων συναίνεσης όπως ο Zookeeper [31] και το LogCabin [32].

Τελικώς, το RAMCloud με όλη την απλότητά του, καταφέρνει να επιλύσει τις σημαντικές προκλήσεις που επιτάσσει η προσπάθεια παροχής ενός συστήματος αποθήκευσης με το μικρότερο δυνατό

χρόνο απόκρισης σε αιτήματα χρηστών. Είναι ένα σύστημα το οποίο επίσης εκμεταλλεύτηκε τη σύγχρονη τάση ανάπτυξης καινοτόμων ιδεών στο χώρο των δικτύων, οπότε και στηρίζεται στην πολύ ταχύτερη πρόσβαση σε μνήμη πάνω από δίκτυο συγκριτικά με προσβάσεις σε δίσκο.

3.4 Hotplugging Μνήμης

Μια βασική πρόκληση στο σχεδιασμό του FluidMem αποτέλεσε η επιλογή της κατάλληλης διεπαφής μέσω της οποίας η μνήμη που θα παρέχεται σε ένα εικονικό μηχάνημα να ελέγχεται από την monitor διεργασία. Αναλυτικότερα, θα θέλαμε ο επιβλέποντας του εικονικού μηχανήματος να αναθέτει σε αυτό μνήμη χρησιμοποιώντας παράλληλα τις συναρτήσεις του libuserfault-client.c αρχείου. Με αυτό τον τρόπο, περιοχές μνήμης που έχουν ανατεθεί στο εικονικό μηχάνημα θα αναπαρίστανται από περιγραφείς αρχείων που θα ελέγχει η monitor διεργασία, ακριβώς σαν να ήταν μια απλή εφαρμογή. Από την έκδοση 2.1 και έπειτα του Qemu, υπάρχει η υποστήριξη για hotplugging μνήμης. Πρόκειται για μια λειτουργικότητα ιδιαίτερα χρήσιμη για τις ανάγκες του FluidMem και την απόδοση μέσω αυτής απομακρυσμένης μνήμης στα εικονικά μηχανήματα.

Με το hotplugging μνήμης παρέχεται η δυνατότητα αύξησης της μνήμης που ήδη ανήκει σε ένα εικονικό μηχάνημα χωρίς να χρειαστεί καταστολή της λειτουργίας του. Από την έκδοση 2.4 και έπειτα του Qemu υλοποιήθηκε επίσης η δυνατότητα για hot-unplugging μνήμης, συνιστώντας έτσι μια εναλλακτική του memory ballooning. Ειδικότερα, ο hypervisor έχει την δύναμη να αυξάνει και να μειώνει τη μνήμη που ανατίθεται στο εικονικό μηχάνημα έπειτα από επίβλεψη των αναγκών του με χρήση των συγκεκριμένων διεπαφών. Σημειώνεται ότι για να είναι δυνατές αυτές οι λειτουργίες, η συνεργασία του φιλοξενούμενου εικονικού μηχανήματος είναι απαραίτητη.

Το FluidMem τελικώς επέλεξε να χρησιμοποιήσει και να επεκτείνει τη δυνατότητα για hotplugging μνήμης ώστε η μνήμη που θα ανατεθεί στο εικονικό μηχάνημα να ελέγχεται από την monitor διεργασία. Έτσι, το FluidMem κάνει χρήση του Qemu hypervisor, τον οποίο επεκτείνει με δυνατότητα ένταξης ενός νέου αντικειμένου μνήμης, που αναπαριστά την ελαστική μνήμη του FluidMem. Το αντικείμενο μνήμης μπορεί να επιλεγεί δυναμικά με χρήση της hotplug διεπαφής, ή μπορεί εξαρχής να αποτελέσει την ολική μνήμη του εικονικού μηχανήματος.

Επομένως, οι δυνατές λειτουργίες του FluidMem είναι δύο, αναφορικά με τις μεθόδους που η απομακρυσμένη μνήμη προσφέρεται ως δευτερεύον χώρος αποθήκευσης στα εικονικά μηχανήματα:

- Μέρος της μνήμης του εικονικού μηχανήματος ελέγχεται από την monitor
- Το σύνολο της μνήμης του εικονικού μηχανήματος ελέγχεται από την monitor

Έτσι, το FluidMem μπορεί να ελέγχει τη μνήμη των εικονικών μηχανημάτων καθιστώντας την ικανή να βρίσκεται ακόμη και ολοκληρωτικά σε απομακρυσμένους κόμβους. Εντός του φιλοξενούμενου εικονικού μηχανήματος, η εικόνα είναι ένας ενιαίος χώρος από μνήμη, η οποία ενδέχεται όμως στην πραγματικότητα όλη ή και μέρος της να ελέγχεται από την monitor διεργασία. Έτσι, προσβάσεις σε αυτή τη μνήμη από εφαρμογές του φιλοξενούμενου εικονικού μηχανήματος που καταλήγουν σε λάθη σελίδας ικανοποιούνται από την monitor διεργασία φέρνοντας την απαραίτητη σελίδα στην τοπική μνήμη και καθιστώντας την προσβάσιμη από το εικονικό μηχάνημα.

Συμπερασματικά, το hotplugging μνήμης είναι το κλειδί για την παροχή ενός περιβάλλοντος πλήρους διαχωρισμένης μνήμης. Το φιλοξενούμενο εικονικό μηχάνημα δεν έχει καμία επίγνωση του υπολογιστικού κόμβου στον οποίο ανά πάσα στιγμή βρίσκεται μέρος της μνήμης του. Επίσης, το φιλοξενούμενο εικονικό μηχάνημα δεν έχει καν επίγνωση ύπαρξης άλλων υπολογιστικών κόμβων. Η monitor διεργασία που τρέχει στον κόμβο που φιλοξενεί το εικονικό μηχάνημα, είναι και η υπεύθυνη για την παροχή σε αυτό της μνήμης που του έχει υποσχεθεί.

3.5 Συνεισφορά της Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία εντάσσει στο σύστημα FluidMem [3] τη δυνατότητα καταγραφής των αποστάσεων επαναχρησιμοποίησης σε σελίδες που ανιχνεύονται με τον μηχανισμό σελιδοποίησης. Τα αποτελέσματα που λαμβάνονται, στη συνέχεια, διέρχονται από ένα στάδιο επεξεργασίας που οδηγεί στην κατασκευή των MRC καμπυλών. Σημαντικό πλεονέκτημα του συστήματος FluidMem αποτελεί η ομοιόμορφη αντιμετώπιση των σελίδων μνήμης είτε αυτές ανήκουν σε διεργασίες είτε ανήκουν σε εικονικά μηχανήματα. Έτσι, οι εξαγόμενες MRC καμπύλες μπορεί να αφορούν τόσο διεργασίες όσο και εικονικά μηχανήματα. Επεξηγηματικά, λάθη σε σελίδες εντός του εικονικού μηχανήματος σε μέρος της μνήμης που ορίστηκε να επιβλέπει το FluidMem διέρχονται μέσω του μηχανισμού σελιδοποίησης και μπορεί να ανιχνευθούν, ακριβώς με τον ίδιο τρόπο με λάθη σελίδων που λαμβάνουν χώρα σε επίπεδο διεργασιών.

Το FluidMem ως ένα σύστημα που δύναται να παρέχει πλήρη διαχωρισμό της μνήμης (Full Memory Dissaggregation) με λύσεις υλοποιημένες αποκλειστικά σε επίπεδο λογισμικού, μπορεί να επωφεληθεί ιδιαίτερα από την γνώση που αφορά στις MRC καμπύλες των εικονικών μηχανημάτων που φιλοξενεί. Συγκεκριμένα, επακριβή γνώση του μεγέθους συνόλου εργασίας κάθε εικονικού μηχανήματος μπορεί να χρησιμοποιηθεί για την αύξηση ή την μείωση της συνολικής παρεχόμενης μνήμης ανά εικονικό μηχάνημα και ανά χρονική στιγμή. Ειδικότερα, σε επίπεδο πολλών κόμβων, λήψη εύλογων αποφάσεων αναφορικά με το μέγεθος του συνόλου εργασίας των εικονικών μηχανημάτων μπορεί να αξιοποιηθεί ως προς την αφαίρεση μνήμης από ένα εικονικό μηχάνημα και την ανάθεσή της σε κάποιο άλλο αποσκοπώντας στην κατά το δυνατόν αποδοτικότερη χρήση των συνολικά διαθέσιμων πόρων. Λεπτομέρειες για επεκτάσεις του FluidMem ώστε με δυναμικό τρόπο να γίνεται αποτίμηση της γνώσης που προσφέρουν οι MRC καμπύλες παρουσιάζονται στο κεφάλαιο 6.

Ο μηχανισμός σελιδοποίησης του FluidMem κρίθηκε ως κατάλληλη διεπαφή για την ενσωμάτωση της λήψης των αποστάσεων επαναχρησιμοποίησης για αρκετούς λόγους. Αρχικά, εναλλακτικές επιλογές απαιτούν τροποποιήσεις στο επίπεδο του πυρήνα του επιβλέποντα και επομένως είναι συνδεδεμένες στενά με κάποιο συγκεκριμένο επιβλέποντα, γεγονός που εισάγει περιορισμούς σε ένα σύστημα που θα θέλαμε να μπορεί να υποστηρίξει πλήθος hypervisors όπως το FluidMem [4].

Έπειτα, η συγκεκριμένη διεπαφή παρέχει λύση σε ένα σημαντικό πρόβλημα: Ερχόμενο σε αντιδιαστολή με άλλα συστήματα υπολογισμού MRC καμπυλών σε εικονικά μηχανήματα [22], στο FluidMem το μέγεθος της ενεργά χρησιμοποιούμενης ενός εικονικού μηχανήματος μπορεί να εκτιμηθεί ανεξαρτήτως της αρχικής μνήμης που έχει αποδοθεί στο εικονικό μηχάνημα. Συγκεκριμένα, το ελάχιστο μέγεθος της ενεργά χρησιμοποιούμενης μνήμης που μπορεί να παρατηρηθεί καθορίζεται μόνο από τον LRU Buffer. Ειδικότερα, το μέγεθος του Buffer μπορεί να ελαττωθεί έως χωρητικότητα 180 σελίδων, όπου και πάλι το εικονικό μηχάνημα είναι ικανό να δέχεται SSH αιτήματα πριν γίνει υπέρβαση του χρονικού ορίου [4].

Τέλος, το FluidMem είναι σε θέση να αποκαλύψει το πραγματικό μέγεθος του συνόλου εργασίας ενός εικονικού μηχανήματος [3] διότι δεν κάνει διακρίσεις στην αντιμετώπιση σελίδων που ανήκουν στο λειτουργικό σύστημα ή σελίδες αρχείων. Δηλαδή, πλέον υπάρχει δυνατότητα μεταφοράς στην απομακρυσμένη μνήμη των συγκεκριμένων σελίδων. Επομένως οι σελίδες αυτές μπορούν να ανιχνευθούν όταν συμβαίνουν προσβάσεις που οδηγούν σε pagefaults, με τρόπο παρόμοιο όπως οι ανώνυμες σελίδες. Αυτό σημαίνει ότι το WSS που θα προκύψει κατά τη μέθοδό μας, δύναται να δώσει εικόνα για το πλήθος των σελίδων αυτών.

Κεφάλαιο 4

Λεπτομέρειες της επέκτασης του FluidMem

Στο κεφάλαιο αυτό γίνεται επεξήγηση όλων των βημάτων και των αποφάσεων που πάρθηκαν ως προς την υλοποίηση της επέκτασης του συστήματος FluidMem για τον υπολογισμό MRC καμπυλών.

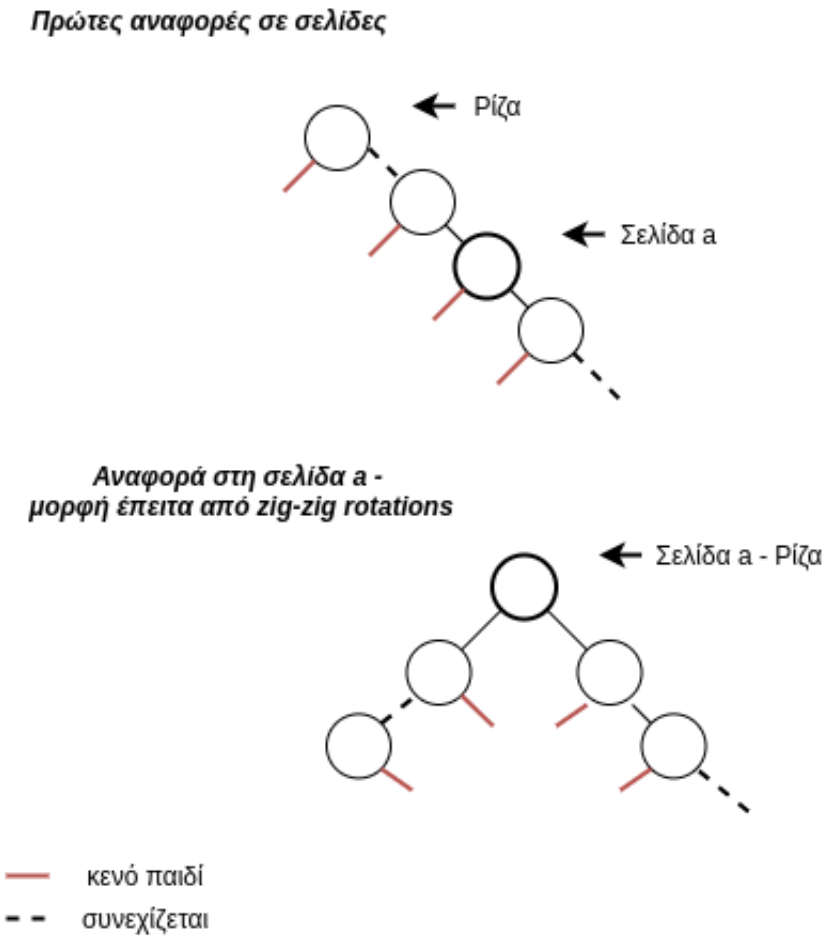
4.1 Σχεδιαστικές αποφάσεις

4.1.1 AVL δέντρα αντί για Splay δέντρα

Η δομή που επιλέχτηκε κατά την εφαρμογή του αλγορίθμου του Olken είναι τα AVL δέντρα έναντι των Splay [33] δέντρων. Τα Splay δέντρα ως πρώτη όψη ενδέχεται να φαίνονται ως μια καλή επιλογή διότι αναπαριστούν αποδοτικά προβλήματα που χαρακτηρίζονται από την αρχή της τοπικότητας. Τα Splay δέντρα είναι αυτορυθμιζόμενα BST δέντρα που βελτιώνουν τα ασυμπτωτικά όρια των BST δέντρων υπό την amortized έννοια [34]. Συγκεκριμένα, η τεχνική του splaying συνίσταται από μια ακολουθία rotations που μετακινούν τον υπό εξέταση κόμβο στη ρίζα του δέντρου, ώστε να ελαχιστοποιηθεί ο χρόνος πρόσβασης σε αυτόν κατά επόμενες αναφορές. Εν γένει, με εφαρμογή της μετακίνησης προς τη ρίζα με συγκεκριμένο τρόπο, κάθε ακολουθία από $M > N$ lookups σε δέντρο κόμβων, απαιτεί $O(MlogN)$ χρόνο. Έτσι, παρατηρούμε βελτίωση συγκριτικά με την $O(MN)$ πολυπλοκότητα που θα έδινε ένα κλασικό BST δέντρο. Κατά τον αλγόριθμο του Olken, όμως, θα αποδειχτεί ότι τα Splay δέντρα δεν δίνουν λογαριθμικό ύψος δέντρου, γεγονός που διασφαλίζεται με χρήση AVL δέντρων σε κάθε περίπτωση.

Ειδικότερα, ο αλγόριθμος του Olken με Splay δέντρα και με γνώμονα τη χρονική αλυσίδα των γεγονότων έχει ως αποτέλεσμα τη διενέργεια rotations ώστε ο νεοεισαχθείς κόμβος να καταλήξει στη ρίζα του δέντρου. Το Splay δέντρο είναι επίσης δέντρο δυαδικής αναζήτησης οπότε δεν υπάρχει διαφορά στον τρόπο αποτίμησης της απόστασης επαναχρησιμοποίησης. Το Splay δέντρο δίνει την επιπλέον ιδιότητα ότι δεν απαιτούνται διαγραφές και επανεντάξεις κόμβων.

Αναλυτικότερα, στις αρχικές προσβάσεις (που βλέπουμε πρώτη φορά) δημιουργείται μια γραμμική αλυσίδα (μέσω της ένταξης σελίδων στο δεξιότερο παιδί). Στην περίπτωση που γίνεται επαναπρόσβαση σε μια σελίδα, πρέπει η σελίδα αυτή να μεταφερθεί στη ρίζα του δέντρου με αναδιάταξη του δέντρου με τρόπο που καθορίζεται από τα Splay δέντρα. Ακριβώς επειδή όλοι οι κόμβοι αποτελούν δεξί παιδί του πατέρα τους, οι αναδιατάξεις αυτές οδηγούν σε μια επίσης γραμμική δόμηση δέντρου. Κατά συνέπεια, στο πρόβλημά μας στο οποίο κόμβοι που αναφέρονται πρώτη φορά εισάγονται πάντα στο δεξιότερο σημείο, το Splay δέντρο θα είναι πάντα μια γραμμική αλυσίδα και επομένως η διάσχισή του θα απαιτεί γραμμικό χρόνο έναντι λογαριθμικού. Έπειτα ακολουθεί σχηματική αναπαράσταση του ανωτέρω επιχειρήματος.



Σχήμα 9: Γραμμικός χρόνος με Splay δέντρα

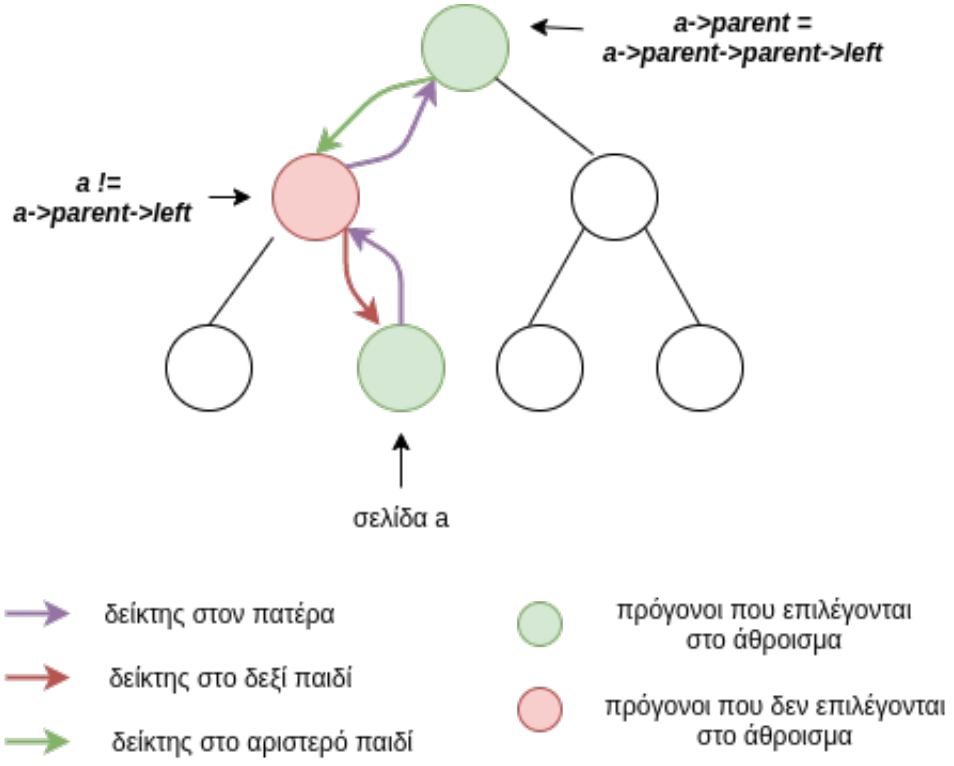
4.1.2 Αποφυγή διατήρησης χρονικού μετρητή

Για τον υπολογισμό της απόστασης στοίβας για μια σελίδα, όπως αναγράφεται και στο τμήμα 2.3.6, πρέπει να εντοπιστεί ο συνολικός αριθμός όλων των προσβάσεων που συνέβησαν πιο πρόσφατα.

Λόγω της ιδιότητας του δυαδικού δέντρου, η απόσταση της στοίβας θα δοθεί με άθροιση όλων των κόμβων που:

- Ανήκουν στο δεξί υποδέντρο του προς εξέταση κόμβου
- Είναι πρόγονοι του προς εξέταση κόμβου και ο κόμβος εντοπίζεται στο αριστερό τους υποδέντρο
- Ανήκουν στο δεξί υποδέντρο όλων των προγόνων του κόμβου που ικανοποιούν την παραπάνω ιδιότητα

Έτσι, προκειμένου να υπολογιστεί η απόσταση στοίβας δεδομένης μιας αναφοράς, πρέπει να γίνει αναζήτηση στο δέντρο από την παρούσα τοποθεσία της αναφοράς και διάσχιση προς την ρίζα του δέντρου. Σε πρώτη φάση η απόσταση στοίβας αρχικοποιείται με το μέγεθος του δεξί υποδέντρου του προς εξέταση κόμβου, μιας και σίγουρα αναπαριστά αναφορές που συνέβησαν μεταγενέστερα χρονικά. Κόμβοι που εντοπίζονται να συμπεριλαμβάνουν στο αριστερό τους υποδέντρο τον κόμβο που μας ενδιαφέρει, είναι πρόγονοι των οποίων τόσο το μέγεθος του δεξιού υποδέντρου τους όσο και οι ίδιοι πρέπει να συνυπολογιστούν στο τελικό άθροισμα. Πιο επεξηγηματικά, αν βρεθεί πρόγονος στου οποίου το αριστερό υποδέντρο ανήκει ο κόμβος που μας ενδιαφέρει, σίγουρα ο ίδιος και οι κόμβοι του δεξιού υποδέντρου του αναπαριστούν αναφορές που συνέβησαν πιο πρόσφατα χρονικά, λόγω της ιδιότητας του δυαδικού δέντρου. Η αναγνώριση του εντοπισμού κόμβων που ικανοποιούν την ιδιότητα ότι περιλαμβάνουν στο αριστερό τους υποδέντρο την αναφερόμενη σελίδα γίνεται με απλή σύγκριση των δεικτών του αριστερού δείκτη του υπό εξέταση προγόνου και του παιδιού του που συναντήσαμε στην πορεία της διάσχισης.



Σχήμα 10: Αλγόριθμος Olken χωρίς timestamps

Σημειώνεται ότι η παραπάνω αναζήτηση θα μπορούσε να γίνει προγραμματιστικά ευκολότερη με διατήρηση ενός χρονικού μετρητή σε κάθε κόμβο, που θα αναπαριστούσε τη χρονική στιγμή που συνέβη μια αναφορά. Έτσι, μια απλή σύγκριση με τη χρονική τιμή του πατέρα θα οδηγούσε σε πιο εύκολη αναγνώριση των κόμβων που πρέπει να συνυπολογιστούν στο άθροισμα, κατά πώς προτείνεται και στο [18].

Η παράκαμψη αυτού του μετρητή, όμως, προσφέρει δύο καθοριστικά οφέλη:

- Οικονομία σε αποθηκευτικό χώρο
- Δεν απαιτείται οριοθέτηση του χρονικού μετρητή σε μια μέγιστη τιμή

Κυρίως σε σχέση με την δεύτερη ιδιότητα, παρακάμπτουμε και προβλήματα που θα πήγαζαν από την ανάγκη επαναρχικοποίησης του μετρητή σε περιπτώσεις υπέρβασης μιας μέγιστης τιμής (overflow).

4.1.3 Διατήρηση μεγέθους MAX_CACHE_SIZE

Με στόχο τον εκ των προτέρων ορισμό ενός άνω φράγματος στο μέγεθος του AVL δέντρου και των μεγίστων αποστάσεων στοίβας που είμαστε σε θέση να ανιχνεύσουμε, επιλέχτηκε η διατήρηση ενός MAX_CACHE_SIZE μεγέθους ως αναπαράσταση του συνολικά μέγιστου ποσού της μνήμης που μπορεί να ανατεθεί ανά εικονικό μηχάνημα. Όταν το AVL δέντρο ξεπεράσει το μέγεθος αυτό, πραγματοποιείται αφαίρεση του κόμβου που εισήχθη παλαιότερα: του κόμβου που εντοπίζεται στο αριστερότερο σημείο. Επαναπροσβάσεις σε σελίδες που αφαιρέθηκαν από το δέντρο λόγω υπέρβασης των αποστάσεων επαναχρησιμοποίησης αντιμετωπίζονται με ίδιο τρόπο όπως πρώτες αναφορές σε αυτές.

4.2 Βοηθητικές δομές δεδομένων

Ακολουθεί ανάλυση των δομών δεδομένων που εντάχθηκαν στην userfaultfd βιβλιοθήκη του FluidMem με στόχο την υποστήριξη της λήψης μετρικών επαναχρησιμοποίησης με τη βοήθεια του αλγόριθμου του Olken. Στα επόμενα, η έννοια της διεργασίας συμπεριλαμβάνει διεργασίες Qemu που σηματοδοτούν την έναρξη των φιλοξενούμενων εικονικών μηχανημάτων.

4.2.1 Σκιαγράφηση της δομής λήψης μετρικών ανά διεργασία

Ανά διεργασία, οι προσθήκες που πρέπει να γίνουν είναι οι εξής:

- **Πεδίο pid**: Η δομή λήψης μετρικών οφείλει να έχει αποθηκευμένη γνώση της διεργασίας που αναπαριστά. Η γνώση προσφέρεται με αποθήκευση του pid μοναδικού αναγνωριστικού -ανά υπολογιστικό κόμβο- της διεργασίας.
- **Πεδίο start_time**: Είναι πιθανό ένας χρήστης των μετρικών επαναχρησιμοποίησης να ενδιαφέρεται για μελέτη στο πλαίσιο ενός χρονικού παραθύρου. Για το λόγο αυτό αποθηκεύεται η εναρκτήρια χρονική στιγμή του αιτήματος λήψης μετρικών, η οποία παρέχεται στον χρήστη ακολουθούμενη από τη χρονική στιγμή του επακόλουθου αιτήματος τέλους.
- **Πεδίο hash_map**: Το κλειδί με βάση το οποίο διατηρείται η ιδιότητα της δυαδικότητας του δέντρου είναι ο χρόνος. Συνεπώς, απαιτείται η αποθήκευση ενός ευρετηρίου ανά διεργασία, που αντιστοιχίζει κάθε σελίδα στην οποία έχει εντοπιστεί μέχρι στιγμής αναφορά με τον κόμβο του AVL δέντρου στον οποίο ανήκει. Το ευρετήριο εντάσσεται σε ευρύτερη δομή ανά διεργασία ως πεδίο με όνομα hash_map. Το πεδίο hash_map είναι τύπου hnode και αποτελείται από ένα πεδίο key, που αντιπροσωπεύει τη διεύθυνση της σελίδας, ένα πεδίο n που αντιπροσωπεύει τον κόμβο του AVL δέντρου και ένα πεδίο hh με ρόλο handler για το ευρετήριο.

- **Πεδίο avl_tree**: Η εισαγωγή νέων κόμβων στο AVL δέντρο είναι εφικτή δεδομένου του δείκτη στη ρίζα του δέντρου που αφορά στις προσβάσεις της υπό μελέτη διεργασίας. Για το σκοπό αυτό διατηρούμε ένα πεδίο avl_tree που αποθηκεύει τον δείκτη που μας ενδιαφέρει.

- **Πεδίο results**: Το συγκεκριμένο πεδίο είναι ένας πίνακας συνολικού μεγέθους MAX_CACHE_SIZE ο οποίος σταδιακά ενημερώνεται με τα αποτελέσματα των αποστάσεων επαναχρησιμοποίησης. Ειδικότερα, εντοπισμός αναφοράς με απόσταση επαναχρησιμοποίησης distance οδηγεί σε αύξηση του αριθμού προσβάσεων results[distance] κατά μια αναφορά. Τα τελικά αποτελέσματα όπως παρέχονται από τον πίνακα results είναι και αυτά που θα επιστραφούν στο χρήστη κατά τη σηματοδότηση λήξης της λήψης μετρικών για τη διεργασία.

- **Πεδίο hh**: Πρόκειται για τον handler του καθολικού ευρετηρίου στο οποίο με κλειδί το αναγνωριστικό pid της διεργασίας εντοπίζεται η ζητούμενη δομή profile_node.

Συνολικά, ανά διεργασία πραγματοποιείται αποθήκευση των εξής πεδίων:

```
struct profile_node {
  uint32_t pid;
  time_t start_time;
  hnode * hash_map;
  AVLTree * avl_tree;
  int * results;
  UT_hash_handle hh;
};
```

4.2.2 Σκιαγράφηση των δομών για το σύνολο των διεργασιών

Σε καθολικό επίπεδο, οι παρακάτω δομές πρέπει να ενταχθούν στην βιβλιοθήκη:

- **Ευρετήριο profileNodes**: Η εύρεση με κλειδί το αναγνωριστικό pid της διεργασίας της δομής profile_node που την αφορά γίνεται με αποδοτικό τρόπο με τη διατήρηση του profileNodes ευρετηρίου. Έτσι, σε χρόνο $O(1)$ είναι εφικτή η ανάκτηση της δομής δοθείσας μιας αναφοράς σε σελίδα της διεργασίας με αναγνωριστικό pid.

- **Κλείδωμα profileNodes_lock**: Το FluidMem είναι ένα πολυνηματικό σύστημα στο οποίο ταυτόχρονα εκτελείται πλήθος εργασιών. Συνθήκες ανταγωνισμού ανάμεσα στα νήματα οφείλουμε να λάβουμε υπ όψιν και στην παρούσα υλοποίηση. Ειδικότερα, η διασφάλιση της πρόσβασης από μοναδικό σημείο τόσο στο profileNodes ευρετήριο όσο και ανά profile_node δομή γίνεται με παροχή του κλειδώματος profileNodes_lock. Συγκεκριμένα, ταυτόχρονα αιτήματα από διαφορετικούς πελάτες δεν μπορούν να οδηγήσουν σε race conditions μιας και ικανοποιούνται με σειριακό τρόπο, όμως θα μπορούσε να επιχειρείται σύγχρονη πρόσβαση κατά την ανίχνευση ενός λάθους σε σελίδα μιας διεργασίας (polling thread) ενώ παράλληλα λαμβάνει χώρα αίτημα λήξης της λήψης των μετρικών για κάποια διεργασία (main thread).

4.3 Ενσωμάτωση στο FluidMem

Στο παρόν τμήμα αναλύεται ο τρόπος ένταξης του υπολογισμού των αποστάσεων επαναχρησιμοποίησης στο FluidMem, συμπεριλαμβανομένης της διεπαφής που παρέχει αιτήματα έναρξης και λήξης της λήψης των μετρικών.

4.3.1 Αίτημα έναρξης λήψης μετρικών

Το αίτημα για έναρξη λήψης μετρικών επαναχρησιμοποίησης στην παρούσα έκδοση του FluidMem πραγματοποιείται με επέκταση του εργαλείου ui με την εντολή profilepid <PID>. Το αίτημα εγγράφεται στο socket στο οποίο ακούει το ui-processing νήμα του FluidMem και το επεξεργάζεται ακολούθως.

Οι δυνατές αποκρίσεις του αιτήματος λήψης μετρικών για μια συγκεκριμένη διεργασία είναι οι εξής:

- Η διεργασία είναι νεκρή
- Η διεργασία δεν είναι εγγεγραμμένη στο FluidMem
- Λαμβάνονται ήδη μετρικές για τη συγκεκριμένη διεργασία
- Η λήψη των μετρικών για τη διεργασία μόλις ξεκίνησε

Όλα τα ανωτέρω ενδεχόμενα εξετάζονται από τη συνάρτηση profilePid(uint32_t pidToProfile) που εισάγεται στην userfaultfd βιβλιοθήκη του FluidMem -userfault.c αρχείο- και καλείται από το ui-processing νήμα. Τα πιθανά ενδεχόμενα αναπαρίστανται με διαφορετικούς κωδικούς επιστροφής της συνάρτησης στο σημείο κλήσης, ώστε να ακολουθήσει ανάλογη ειδοποίηση στο δημιουργό του αιτήματος με εγγραφή της απάντησης στο socket.

Στην περίπτωση σημείωσης επιτυχίας, τα βήματα που ακολουθούνται είναι τα εξής:

- **Αρχικοποίηση μιας νέας δομής profileNode**
  Σε αυτό το βήμα γίνεται εκχώρηση μνήμης και αρχικοποίηση των πεδίων της δομής profileNode που θα αφορά τη διεργασία με αναγνωριστικό pid. Υπεύθυνη συνάρτηση είναι η newProfileNode(uint32_t pid) που εντοπίζεται στο αρχείο userfault.c
- **Εισαγωγή του profileNode στο ευρετήριο profileNodes**
  Στη συνέχεια, η δομή profileNode που μόλις δημιουργήθηκε εντάσσεται στο ευρετήριο profileNodes με αναγνωριστικό pid για μελλοντική ανεύρεση.

4.3.2 Αίτημα λήξης της λήψης μετρικών

Το αίτημα για λήξη της λήψης μετρικών επαναχρησιμοποίησης στην παρούσα έκδοση του FluidMem πραγματοποιείται με επέκταση του ui εργαλείου με την εντολή stopprofilepid <PID>. Το αίτημα εγγράφεται στο socket στο οποίο ακούει το ui-processing νήμα του FluidMem και το επεξεργάζεται ακολούθως, ακριβώς όπως και στην περίπτωση αιτήματος έναρξης.

Οι δυνατές αποκρίσεις του αιτήματος για μια συγκεκριμένη διεργασία είναι οι εξής:

- Δεν έχει προηγηθεί αίτημα έναρξης της λήψης μετρικών για τη διεργασία [1]
- Παράθεση των μετρικών στο χρήστη

[1] Το ενδεχόμενο αυτό μπορεί επίσης να προκύψει στην περίπτωση ενεργοποίησης του reaperthread νήματος και με αποστολή του αιτήματος έπειτα από το τέλος της ζωής της διεργασίας.

Τα ανωτέρω ενδεχόμενα εξετάζονται από τη συνάρτηση stopProfilePid(time_t * startTime, int * results, uint32_t pid) που εισάγεται στο userfault.c αρχείο και καλείται από το ui-processing νήμα. Επιπλέον, τα πιθανά ενδεχόμενα αναπαρίστανται με διαφορετικούς κωδικούς επιστροφής στο σημείο κλήσης.

Στην περίπτωση σημείωσης επιτυχίας λαμβάνουν χώρα τα εξής:

- **Αντιγραφή του results πεδίου του profileNode στον results buffer**
  Η αντιγραφή των αποτελεσμάτων στον results buffer γίνεται γιατί θα ακολουθήσει διαγραφή της δομής profileNode.
- **Αντιγραφή του χρόνου έναρξης start_time στον startTime buffer**
  Ομοίως, γίνεται αντιγραφή του χρόνου έναρξης.
- **Απελευθέρωση της profileNode δομής**
  Όλα τα πεδία της δομής profileNode απελευθερώνονται με τη συνάρτηση deleteProfileNode(struct profile_node * n) και ο χώρος διατίθεται ξανά προς χρήση.

Τελικά, από το επιτυχημένο αίτημα ο χρήστης λαμβάνει δυάδες τύπου $\langle cache\_size, references \rangle$, ≥ 0.

4.3.3 Ενσωμάτωση στο μηχανισμό σελιδοποίησης

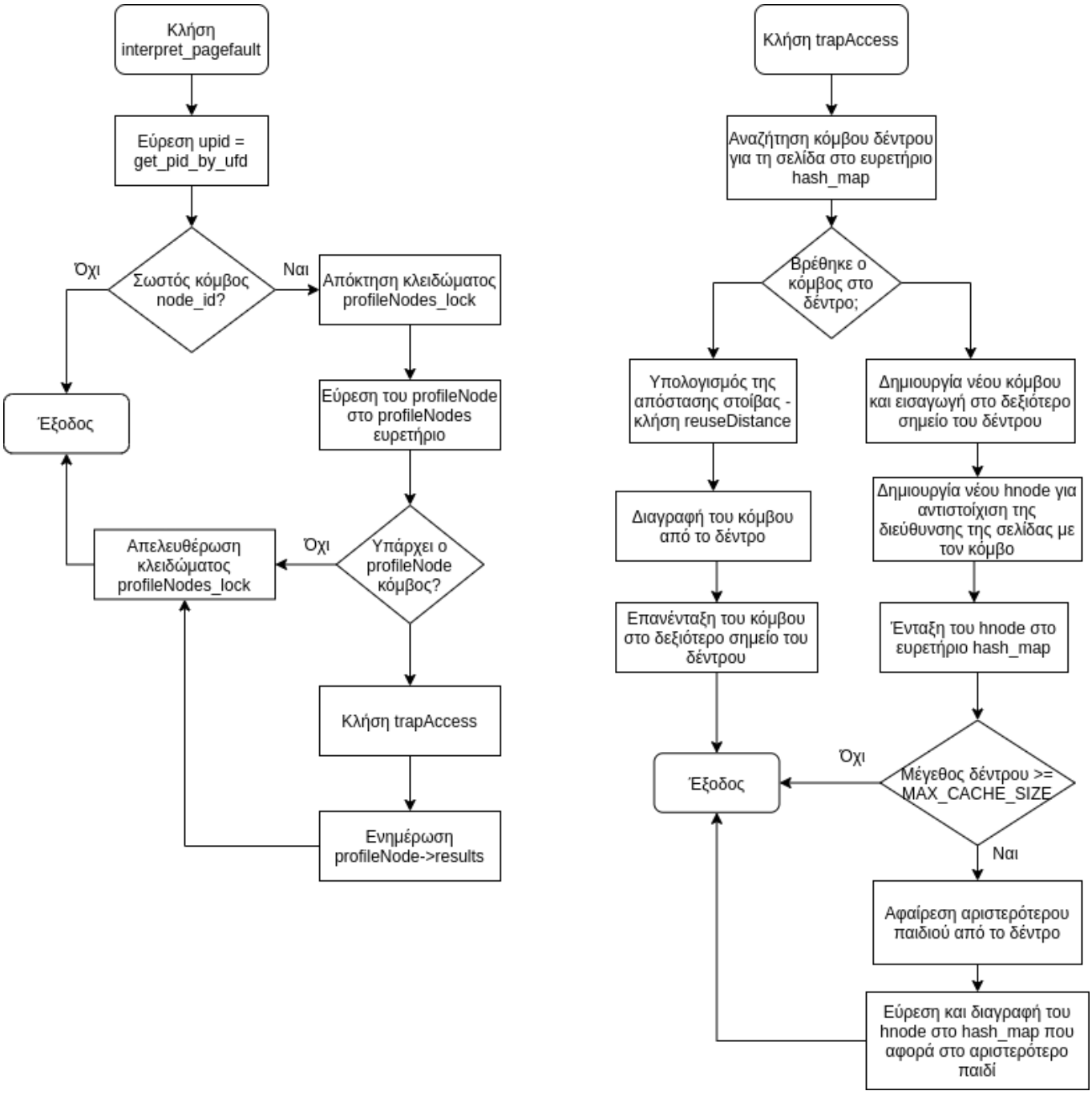
Οι μετρικές επαναχρησιμοποίησης ανανεώνονται κάθε φορά που συναντάται λάθος σε σελίδα διεργασίας για την οποία έχει γίνει αίτημα λήψης των μετρικών. Στη διαδρομή της σελιδοποίησης, η monitor διεργασία συναντάει λάθος σελίδας σε συγκεκριμένο ufd file descriptor. Στην περίπτωση εντοπισμού κάποιου γεγονότος σε οποιονδήποτε από τους εγγεγραμμένους στο FluidMem file descriptors, το polling νήμα καλεί τη συνάρτηση handle_userfault(int ufd) για εξέταση του γεγονότος.

Όταν διαπιστωθεί ότι το γεγονός πρόκειται για pagefault (UFFD_EVENT_PAGEFAULT) και αφού ανακτηθεί η διεύθυνση της σελίδας από το μήνυμα του γεγονότος, εντάσσουμε κλήση στην interpret_pagefault(int ufd, uint64_t pageaddr) του userfault.c αρχείου.

Στη συνέχεια, τα βήματα που ακολουθούνται είναι τα εξής:

- **Εύρεση του pid μέσω του ufd**
  Για το βήμα αυτό αξιοποιείται η ήδη υπάρχουσα συνάρτηση get_upid_by_fd(ufd) - αρχείο include/upid.h του FluidMem που ανακτά το PID αναγνωριστικό της διεργασίας από το ευρετήριο fdUpidMap σε $O(1)$ χρόνο.
- **Εύρεση του profileNode κόμβου στο profileNodes ευρετήριο**
  Έπειτα εκτελείται αναζήτηση της δομής profileNode για το συγκεκριμένο αναγνωριστικό pid στο ευρετήριο profileNodes. Εύρεσή του υποδηλώνει ότι προηγήθηκε αίτημα έναρξης λήψης μετρικών, ενώ μη-εύρεσή του σημαίνει το αντίθετο, στην οποία περίπτωση και τερματίζει η συνάρτηση.
- **Υπολογισμός της απόστασης επαναχρησιμοποίησης**
  Στο βήμα αυτό γίνεται κλήση της συνάρτησης trapAccess(hnode ** records, AVLTree * t, uint64_t addr) που ανήκει στο αρχείο include/reuse_distance.c. Σε αυτό το σημείο λαμβάνει χώρα η εύρεση της σελίδας στο δέντρο, ο υπολογισμός της απόστασης στοίβας με τον αλγόριθμο του Olken, η διαγραφή και η επανεισαγωγή της στην κορυφή της στοίβας ακολοθούμενη από την ανανέωση του δείκτη του ευρετηρίου.
- **Ανανέωση του results πεδίου του profileNode**
  Βάσει της απόστασης επαναχρησιμοποίησης που λάβαμε, γίνεται αύξηση κατά μια αναφορά στην αντίστοιχη θέση του πίνακα profileNode->results.

Τα παρακάτω παρουσιάζονται με περισσότερη λεπτομέρεια στα διαγράμματα ροής που ακολουθούν, για τις συναρτήσεις interpret_pagefault και trapAccess.



Σχήμα 11: Διαγράμματα ροής για την interpret_pagefault και την trapAccess

Κεφάλαιο 5

Πειραματική Αξιολόγηση

Στο παρόν κεφάλαιο πραγματοποιείται καταγραφή και αξιολόγηση των πειραμάτων που αφορούν στον αλγόριθμο εξαγωγής του Ιστογράμματος Απόστασης Επαναχρησιμοποίησης και της Miss Ratio Καμπύλης. Αναλυτικότερα, αρχικά περιγράφεται το περιβάλλον διεξαγωγής των πειραμάτων και έπειτα παρατίθενται τα διαφορετικά σενάρια που υλοποιήθηκαν συνοδευόμενα από τα αποτελέσματά τους.

5.1 Περιβάλλον διεξαγωγής

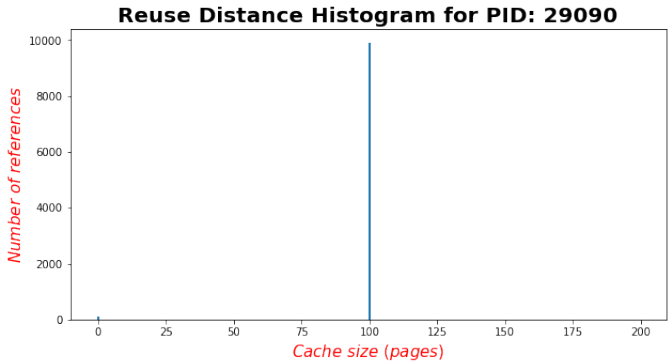
Όλα τα πειράματα υλοποιήθηκαν σε εικονικό μηχάνημα συνολικής μνήμης 8GB με πυρήνα Linux 4.20.0-rc7 στον οποίο ενσωματώθηκαν τα remap patches του Andrea Arcangeli [28]. Στο ίδιο σύστημα τρέχει επιπλέον ένα στιγμιότυπο Zookeeper, για το ρόλο του γενικού συντονιστή του RAMCloud αλλά και για το ίδιο το FluidMem, για την διαχείριση της γνώσης των διεργασιών που έχει αναλάβει η monitor διεργασία. Στα πειράματά μας επιλέξαμε το RAMCloud σύστημα ως το key-value store. Μια διεργασία coordinator τρέχει τοπικά στο σύστημα, ενώ μια διεργασία server τρέχει σε ξεχωριστό εικονικό μηχάνημα, προσπελάσιμο πάνω τοπικό δίκτυο (local network). Συνολικά, 2GB μνήμης προσφέρονται μέσω του server στο RAMCloud δίκτυο.

5.1.1 Πρώτο Σενάριο

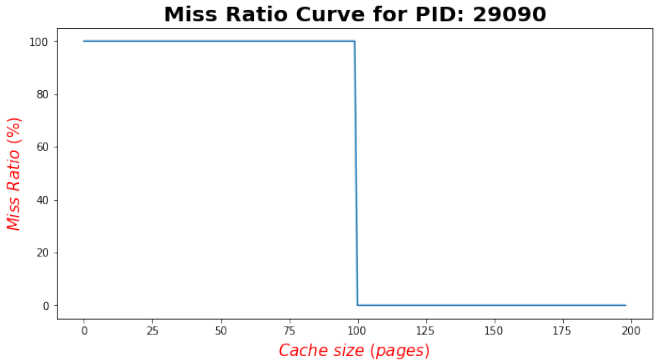
Στόχος του συγκεκριμένου σεναρίου αποτελεί η εξακρίβωση της ορθότητας του αλγορίθμου για μια εφαρμογή με πολύ απλό αποτύπωμα μνήμης. Δοκιμάζουμε τον αλγόριθμο με μια εφαρμογή που εκτελεί κυκλικά προσβάσεις σε δεδομένο αριθμό σελίδων. Ορίσαμε αριθμό σελίδων pages = 100 και αριθμό κύκλων cycles = 100. Θέτουμε αρχικά τον LRU Buffer με μέγεθος 1. Επίσης, το μέγιστο δυνατό μέγεθος της μνήμης ορίστηκε στις 200 σελίδες, δεδομένου ότι με 100 διαφορετικές σελίδες δεν θα εμφανιστεί απόσταση επαναχρησιμοποίησης μεγαλύτερη των 100 σελίδων.

Ως προς τα αποτελέσματα, αναμένουμε οι 100 πρώτες προσβάσεις να εμφανίσουν απόσταση επαναχρησιμοποίησης άπειρη, την οποία αναπαριστούμε με 0. Έπειτα, λόγω των κυκλικών συνεχών προσβάσεων, αναμένουμε όλες οι υπόλοιπες προσβάσεις να δίνουν απόσταση επαναχρησιμοποίησης 100. Συνολικά, αναμένουμε $100 * 99$ προσβάσεις με απόσταση επαναχρησιμοποίησης ίση με 100, και καμία άλλη εμφάνιση πρόσβασης σε άλλες αποστάσεις.

Πράγματι, όπως φαίνεται από τα παρακάτω διαγράμματα τα αποτελέσματα επιβεβαιώνουν την ιδέα, με 9900 προσβάσεις με απόσταση 100 και 100 προσβάσεις με απόσταση 0.

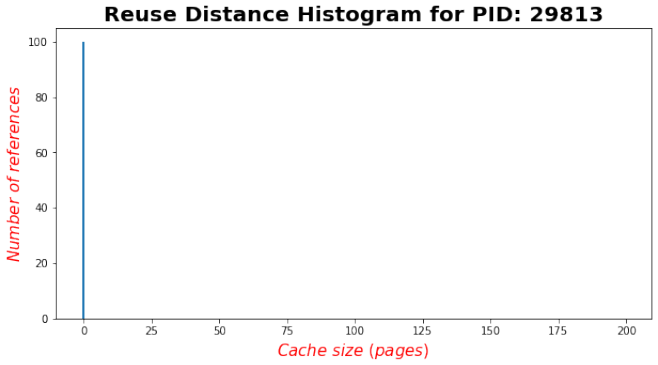


Σχήμα 12: Reuse Distance Histogram, 100 Pages, 100 Cycles, LRU 1



Σχήμα 13: Miss Ratio Curve, 100 Pages, 100 Cycles, LRU 1

Έπειτα, το πείραμα επαναλαμβάνεται με αύξηση του μεγέθους του Buffer, και τα αποτελέσματα προκύπτουν ακριβώς τα ίδια για όλα τα μεγέθη μέχρι και μέγεθος 100, όπου όλες οι σελίδες χωράνε στον Buffer και πλέον όλες οι προσβάσεις καταλήγουν σε επιτυχίες, οπότε είναι μη ανιχνεύσιμες από τον μηχανισμό σελιδοποίησης.



Σχήμα 14: Reuse Distance Histogram, 100 Pages, 100 Cycles, LRU 100

Τα παραπάνω είναι αναμενόμενα αφού για μέγεθος Buffer μικρότερο του 100, οι σειριακές προσβάσεις έχουν ως αποτέλεσμα να γίνεται πρόσβαση πάντα σελίδες που είναι εκτός Buffer, ενώ στην οριακή περίπτωση με μέγεθος Buffer 99 γίνεται πάντα πρόσβαση στην μοναδική σελίδα που μόλις μεταφέρθηκε στην απομακρυσμένη μνήμη. Αυτός είναι ο λόγος που όλες οι προσβάσεις καταλήγουν

σε αποτυχίες στην περίπτωση αυτή.

5.1.2 Δεύτερο Σενάριο

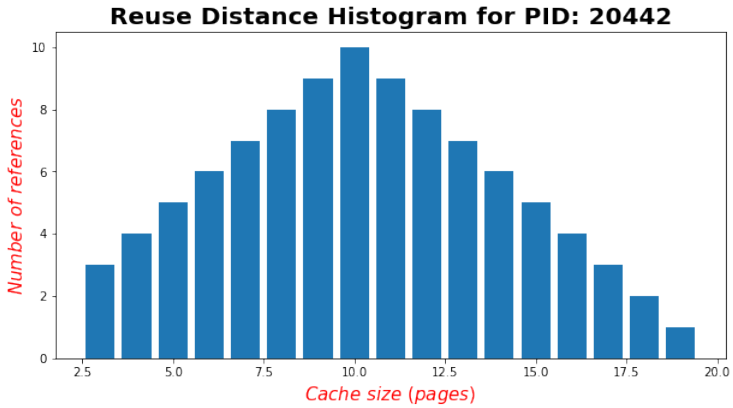
Στόχος του συγκεκριμένου σεναρίου αποτελεί η διερεύνηση των αστοχιών που θα ανιχνευθούν για ένα microbenchmark που εκτελεί προσβάσεις με ομοιόμορφο τρόπο. Το benchmark που τρέξαμε είναι φτιαγμένο με τρόπο ώστε αν υποθέσουμε μέγεθος LRU Buffer μιας σελίδας και άνω, να παράγεται πίνακας αποτελεσμάτων για μετρικές επαναχρησιμοποίησης τύπου:

[0, 2, 3, 4, ... MAX, MAX-1, MAX-2, ..., 1]

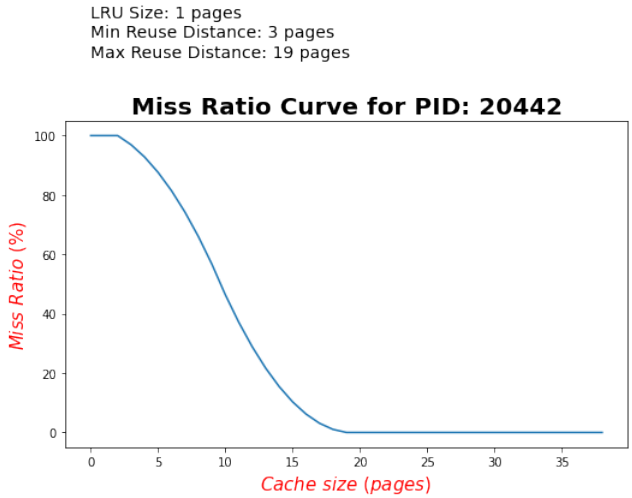
όπου η πρώτη θέση αντιστοιχεί σε απόσταση επαναχρησιμοποίησης 1 -την οποία έτσι και αλλιώς δεδομένου του Buffer δεν μπορούμε να πιάσουμε- και MAX μεταβλητή που δίνεται ως είσοδος. Συνολικά θα εκτελεστούν MAX^2-1 προσβάσεις.

Αρχικά ορίσαμε MAX=10 , LRU Buffer = 1 και MAX_CACHE_SIZE=40 σελίδες.

Λάβαμε τα παρακάτω αποτελέσματα:



Σχήμα 15: Reuse Distance Histogram, MAX=10, Pages=99, LRU 1

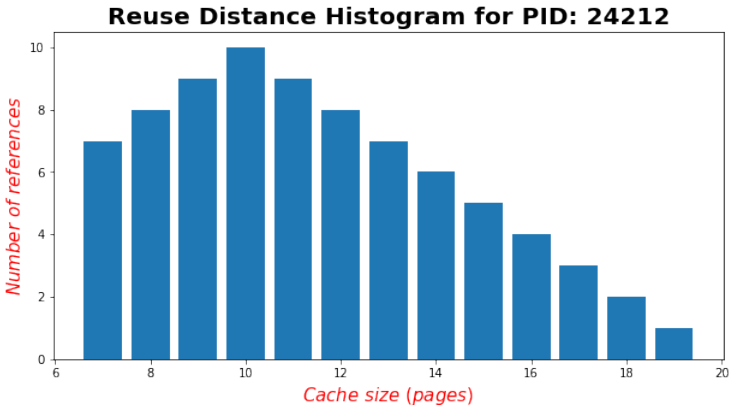


Σχήμα 16: Miss Ratio Curve, MAX=10, Pages=99, LRU 1

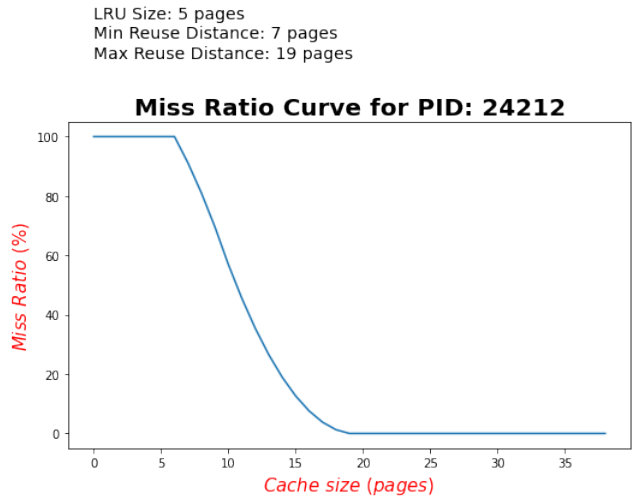
Από τα αποτελέσματα που λάβαμε είδαμε ότι για μέγεθος LRU 1 μπορούμε να ανιχνεύσουμε όλες τις προσβάσεις που απέχουν απόσταση επαναχρησιμοποίησης 3 και άνω. Το αναμενόμενο αποτέλεσμα θα ήταν η ανίχνευση όλων των σελίδων με απόσταση επαναχρησιμοποίησης 2 και άνω. Επομένως, σελίδες που απέχουν απόσταση επαναχρησιμοποίησης 2 δεν τις λαμβάνουμε στο τελικό αποτέλεσμα. Αυτό συμβαίνει διότι η διεργασία που εκτελεί τις προσβάσεις προλαβαίνει να εκτελέσει

hit στη σελίδα που πρόκειται να γίνει remapped πριν συμβεί πράγματι το remapping (page eviction). Εξαιτίας του συγκεκριμένου φαινομένου συγχρονισμού, συνεπώς, δεν είναι δυνατή η ανίχνευση σελίδων που απέχουν απόσταση στοίβας 2.

Έπειτα ορίσαμε MAX=10 , LRU Buffer = 5 και MAX_CACHE_SIZE=40 σελίδες.



Σχήμα 17: Reuse Distance Histogram, MAX=10, Pages=99, LRU 5



Σχήμα 18: Miss Ratio Curve, MAX=10, Pages=99, LRU 5

Σε αυτή την περίπτωση βλέπουμε επίσης ότι η ελάχιστη απόσταση επαναχρησιμοποίησης που λαμβάνουμε είναι 7, ενώ θεωρητικά αναμένουμε ανίχνευση από μεγέθη 6 και άνω. Αυτό εξηγείται ακριβώς όπως στην περίπτωση της απόστασης επαναχρησιμοποίησης 2 για μέγεθος Buffer 1, μιας και στη σελίδα που πρόκειται να γίνει remapped όταν υπάρχει υπέρβαση στο μέγεθος του Buffer γίνεται hit πριν γίνει το remapping από την αρχική virtual memory area.

5.1.3 Τρίτο Σενάριο

Στόχος του συγκεκριμένου σεναρίου είναι η εκτέλεση πειραμάτων για την περίπτωση απλών διεργασιών και η λήψη συμπερασμάτων ως προς το overhead. Τα πειράματα εκτελούνται με χρήση ενός custom microbenchmark τυχαίων προσβάσεων σε μνήμη που αρχικοποιήθηκε με sequential τρόπο κατά την έναρξή του. Το microbenchmark λαμβάνει τις εξής παραμέτρους εισόδου:

- Τον αριθμό των σελίδων μνήμης (WSS)
- Τον αριθμό των προσβάσεων που θα εκτελεστούν σε αυτές

Το πείραμα εκτελέστηκε με DRAM-backed FluidMem (αποτρέπεται και το network overhead που απαιτείται για επικοινωνία με την απομακρυσμένη μνήμη), και με ενεργοποιημένους τους μηχανισμούς βελτιστοποίησης του FluidMem για επιπλέον ελαχιστοποίηση του χρόνου. Η σύγκριση αφορά το overhead που προκύπτει στην περίπτωση ενεργοποίησης της λήψης μετρικών και μη. Για κάθε μέγεθος WSS που επιλέχτηκε να μελετηθεί τα πειράματα πραγματοποιήθηκαν 5 φορές, τόσο για την περίπτωση της ενεργοποίησης της λήψης μετρικών όσο και για την native περίπτωση. Επίσης παρατίθενται τα διαγράμματα του Ιστογράμματος της Απόστασης Επαναχρησιμοποίησης και η MRC καμπύλη για κάθε ένα από τα μεγέθη ενεργά χρησιμοποιούμενης μνήμης, ενδεικτικά.

Σταθεροί παράμετροι του πειράματος αποτελούν:

- LRU capacity 10MB
- Αριθμός προσβάσεων 1.000.000

Τα αποτελέσματα που προέκυψαν για μεγέθη WSS 20MB, 30MB, 40MB και 50MB είναι ακολούθως:

Διεργασία ενεργά χρησιμοποιούμενης μνήμης 20MB, αριθμός προσβάσεων 1.000.000, LRU Buffer μεγέθους 10MB

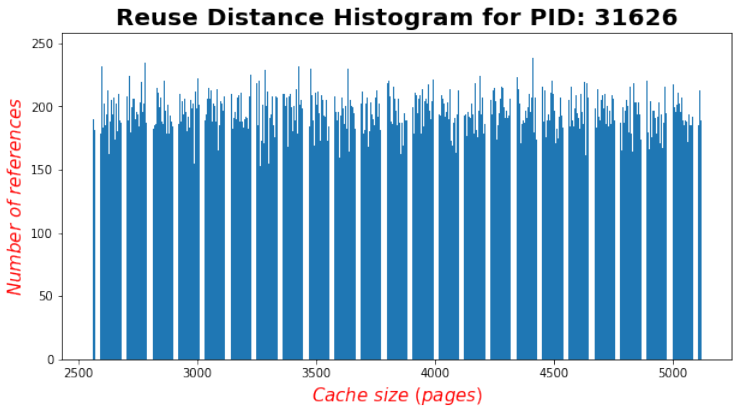
Πίνακας 1: Χρόνος εκτέλεσης αρχικοποίησης (δευτερόλεπτα)

Κατηγορία πειράματος	Πρώτο	Δεύτερο	Τρίτο	Τέταρτο	Πέμπτο	Μέσος όρος
Με λήψη μετρικών	0.2651	0.2464	1.1845	0.1634	0.1366	0.3992
Χωρίς λήψη μετρικών	1.1845	0.3480	0.2406	0.2332	0.2193	0.4451

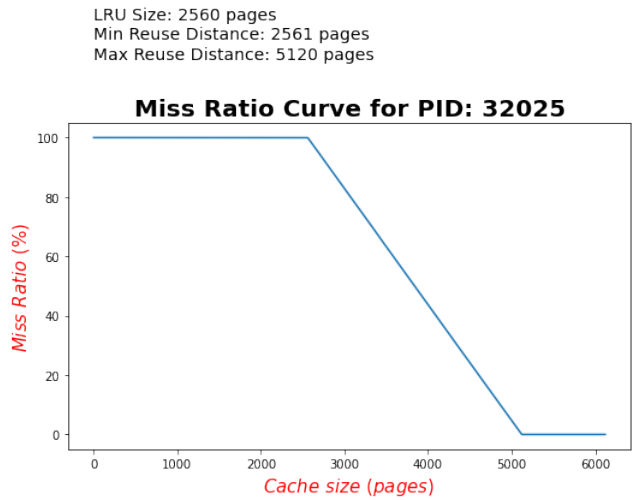
Πίνακας 2: Χρόνος εκτέλεσης κύριου πειράματος (δευτερόλεπτα)

Κατηγορία πειράματος	Πρώτο	Δεύτερο	Τρίτο	Τέταρτο	Πέμπτο	Μέσος όρος
Με λήψη μετρικών	27.6795	27.3370	27.6202	24.3091	28.1845	27.026
Χωρίς λήψη μετρικών	30.7465	31.1845	30.6896	25.1845	27.4843	29.058

Initialization overhead: -10% Accesses overhead: -6.99%



Σχήμα 19: Reuse Distance Histogram, LRU 10MB, WSS 20MB, 1000000 accesses



Σχήμα 20: Miss Ratio Curve, LRU 10MB, WSS 20MB, 1000000 accesses

Διεργασία ενεργά χρησιμοποιούμενης μνήμης 30MB, αριθμός προσβάσεων 1.000.000, LRU Buffer μεγέθους 10MB

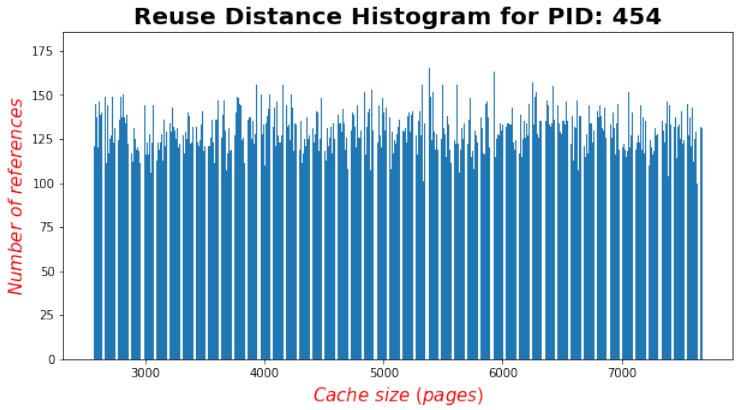
Πίνακας 3: Χρόνος εκτέλεσης αρχικοποίησης (δευτερόλεπτα)

Κατηγορία πειράματος	Πρώτο	Δεύτερο	Τρίτο	Τέταρτο	Πέμπτο	Μέσος όρος
Με λήψη μετρικών	0.2919	1.1845	0.4730	1.1845	0.4657	0.7199
Χωρίς λήψη μετρικών	0.4966	0.4718	0.4326	0.2876	0.3017	0.398

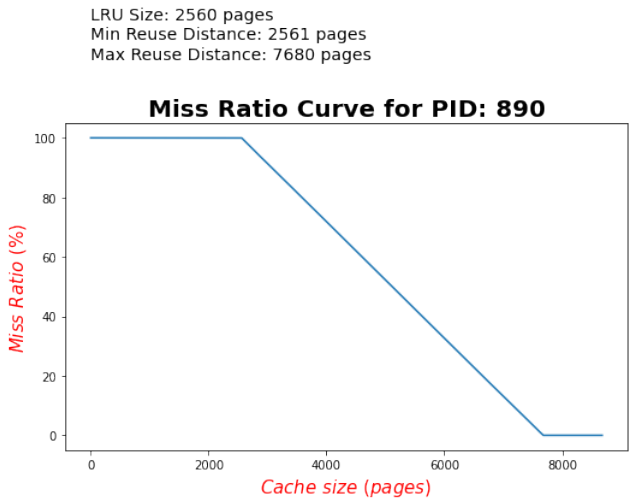
Πίνακας 4: Χρόνος εκτέλεσης κύριου πειράματος (δευτερόλεπτα)

Κατηγορία πειράματος	Πρώτο	Δεύτερο	Τρίτο	Τέταρτο	Πέμπτο	Μέσος όρος
Με λήψη μετρικών	28.1845	42.6137	43.1845	43.9917	43.1653	40.2279
Χωρίς λήψη μετρικών	43.1845	42.1845	42.1845	32.1845	25.1845	36.9845

Initialization overhead: 80.9% Accesses overhead: 8.77%



Σχήμα 21: Reuse Distance Histogram, LRU 10MB, WSS 30MB, 1000000 accesses



Σχήμα 22: Miss Ratio Curve, LRU 10MB, WSS 30MB, 1000000 accesses

Διεργασία ενεργά χρησιμοποιούμενης μνήμης 40MB, αριθμός προσβάσεων 1.000.000, LRU Buffer μεγέθους 10MB

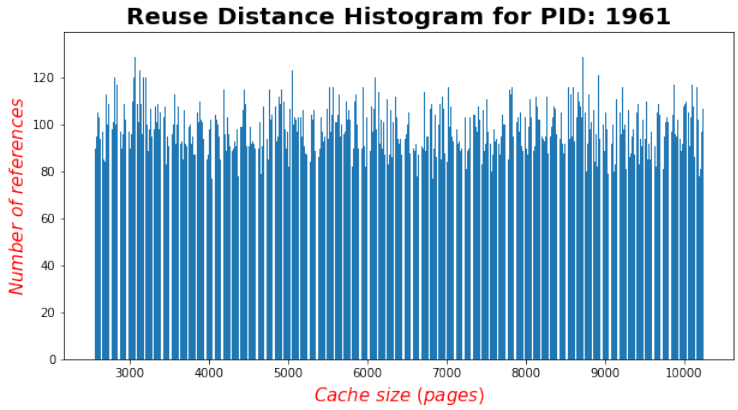
Πίνακας 5: Χρόνος εκτέλεσης αρχικοποίησης (δευτερόλεπτα)

Κατηγορία πειράματος	Πρώτο	Δεύτερο	Τρίτο	Τέταρτο	Πέμπτο	Μέσος όρος
Με λήψη μετρικών	0.6719	1.1845	0.664	0.6326	1.1845	0.8675
Χωρίς λήψη μετρικών	1.1845	0.6685	1.1845	1.1845	0.6677	0.9779

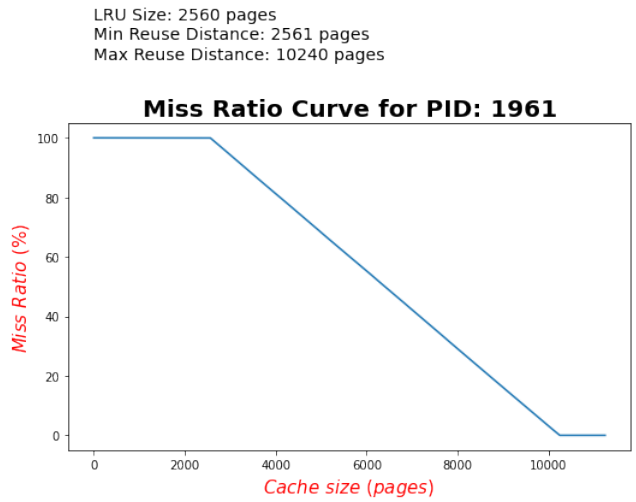
Πίνακας 6: Χρόνος εκτέλεσης κύριου πειράματος (δευτερόλεπτα)

Κατηγορία πειράματος	Πρώτο	Δεύτερο	Τρίτο	Τέταρτο	Πέμπτο	Μέσος όρος
Με λήψη μετρικών	50.1845	48.6028	52.1845	50.1845	49.2692	50.0851
Χωρίς λήψη μετρικών	51.1845	50.1845	47.1845	48.1845	47.1845	48.7845

Initialization overhead: -11.3% Accesses overhead: 2.66%



Σχήμα 23: Reuse Distance Histogram, LRU 10MB, WSS 40MB, 1000000 accesses



Σχήμα 24: Miss Ratio Curve, LRU 10MB, WSS 40MB, 1000000 accesses

Διεργασία ενεργά χρησιμοποιούμενης μνήμης 50MB, αριθμός προσβάσεων 1.000.000, LRU Buffer μεγέθους 10MB

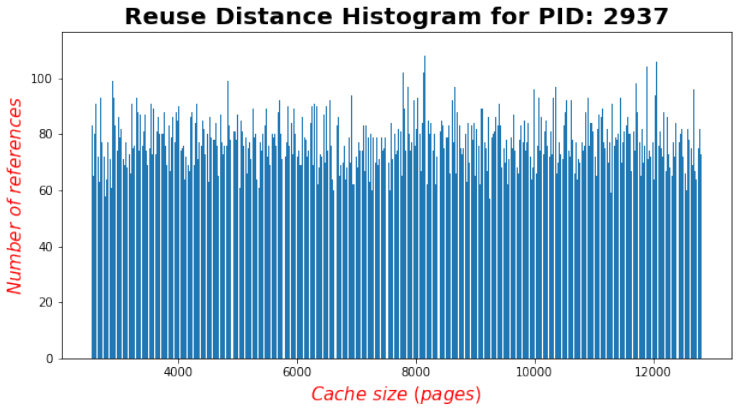
Πίνακας 7: Χρόνος εκτέλεσης αρχικοποίησης (δευτερόλεπτα)

Κατηγορία πειράματος	Πρώτο	Δεύτερο	Τρίτο	Τέταρτο	Πέμπτο	Μέσος όρος
Με λήψη μετρικών	1.1845	1.1845	1.1845	1.1845	1.1845	1.1845
Χωρίς λήψη μετρικών	1.1845	1.1845	0.4804	1.1845	1.1845	1.044

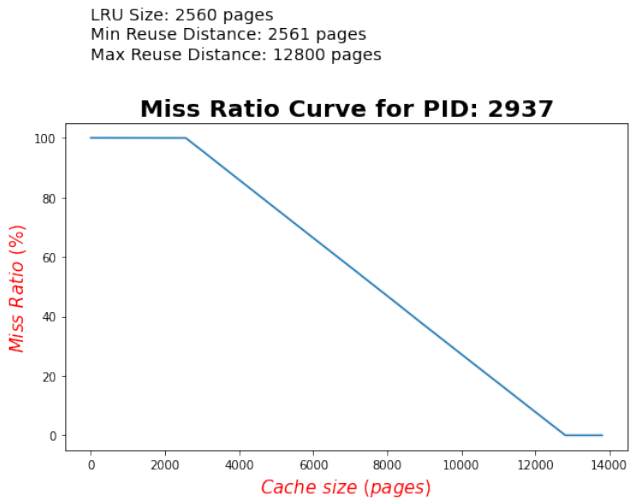
Πίνακας 8: Χρόνος εκτέλεσης κύριου πειράματος (δευτερόλεπτα)

Κατηγορία πειράματος	Πρώτο	Δεύτερο	Τρίτο	Τέταρτο	Πέμπτο	Μέσος όρος
Με λήψη μετρικών	52.3430	52.5332	52.4235	52.1845	55.81527	53.059
Χωρίς λήψη μετρικών	53.1798	51.1325	52.1845	51.1845	50.1845	51.57316

Initialization overhead: 13.5% Accesses overhead: 2.88%

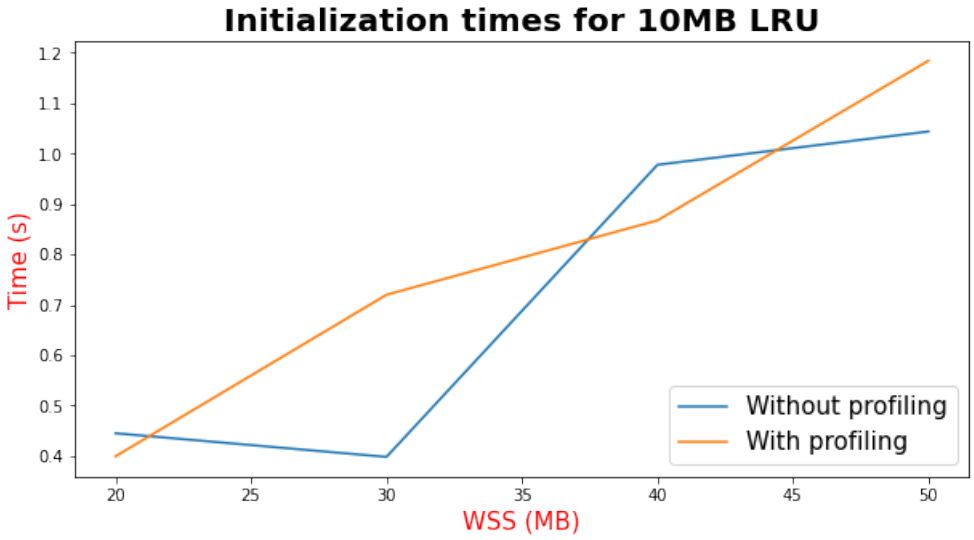


Σχήμα 25: Reuse Distance Histogram, LRU 10MB, WSS 50MB, 1000000 accesses

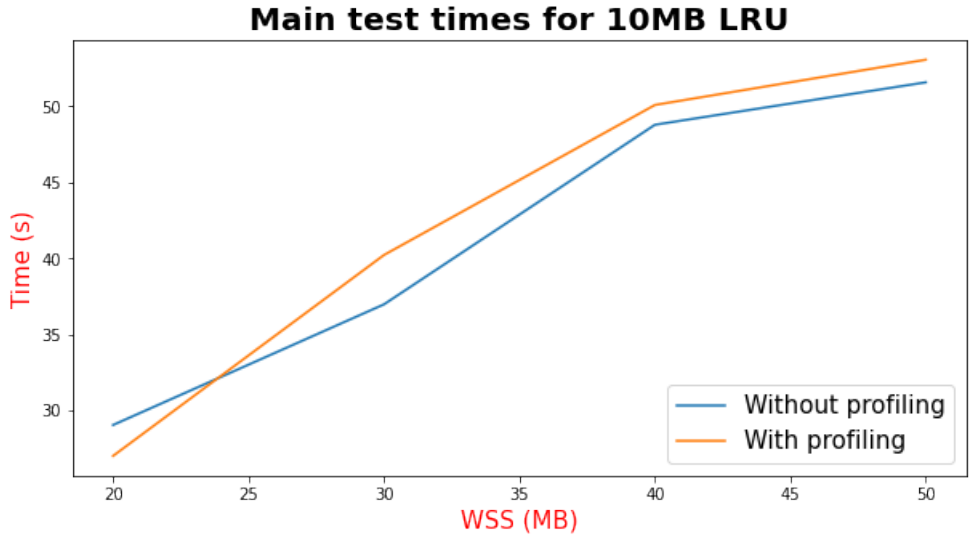


Σχήμα 26: Miss Ratio Curve, LRU 10MB, WSS 50MB, 1000000 accesses

Επίσης, παρουσιάζονται τα διαγράμματα του χρόνου συναρτήσει του μεγέθους της ενεργά χρησιμοποιούμενης μνήμης για σταθερό μέγεθος LRU Buffer 10MB.



Σχήμα 27: Initialization times per WSS, 10MB LRU, 1.000.000 accesses



Σχήμα 28: Main test times per WSS, 10MB LRU, 1.000.000 accesses

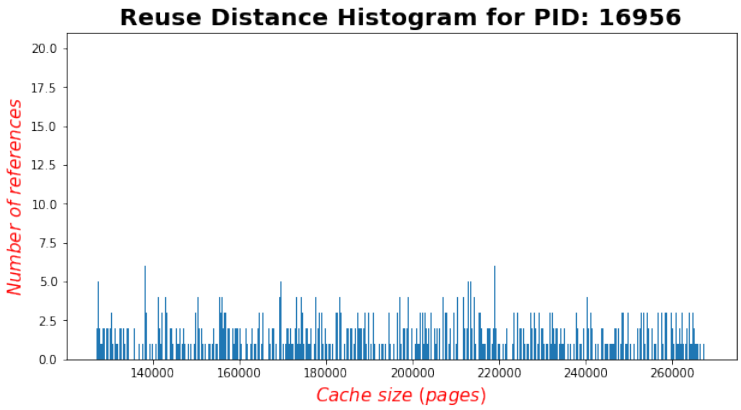
5.1.4 Τέταρτο Σενάριο

Στο σενάριο αυτό δοκιμάζουμε εικονικά μηχανήματα αντί για διεργασίες στα οποία τρέχει το pmbench microbenchmark [35], διαμορφωμένο έτσι ώστε να εκτελεί αρχικά πέρασμα σε ένα σύνολο της μνήμης με τυχαίες αρχικοποιήσεις και εν συνεχεία να εκτελεί προσβάσεις με τυχαίο τρόπο στις αρχικοποιημένες σελίδες, με λόγο γραψίματος - διαβάσματος 50% σε διάρκεια 100, 200 και 1000 δευτερολέπτων. Όλες οι δοκιμές αφορούν εικονικά μηχανήματα των οποίων το σύνολο της μνήμης που τους αποδόθηκε αφορά σε μνήμη του FluidMem. Επίσης, το μέγεθος του LRU Buffer τίθεται σε μέγεθος μικρότερο από την ενεργά χρησιμοποιούμενη μνήμη της εφαρμογής, ώστε να επιφέρει λάθη σελίδων και εκτίμηση του πραγματικού μεγέθους του συνόλου εργασίας μέσα από τις MRC καμπύλες. Για το σενάριο αυτό, το MAX_CACHE_SIZE τέθηκε στα 4GB συνολικά. Επίσης συμπεριλήφθηκαν οι μηχανισμοί βελτιστοποίησης του FluidMem, μιας και μας απασχολεί μια κατά το δυνατόν πιο αποδοτική μορφή της για τα tests. Άλλωστε, το overhead που θα εκτιμήσουμε αφορά στην προσθήκη του αλγορίθμου του Olken στη διαδρομή του userfault handling.

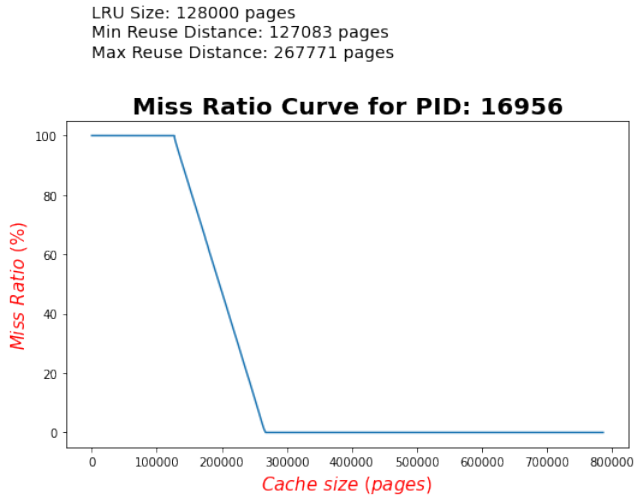
Μετρήσαμε το overhead που εισάγει ο αλγόριθμος όταν τρέχει για 100, 200 και 1000 δευτερόλεπτα αντιστοίχως. Το overhead προέκυψε έπειτα από σύγκριση του μέσου χρόνου ανά πρόσβαση όπως παρατηρήθηκε κατά το διάστημα αυτό.

Εικονικό μηχάνημα συνολικής μνήμης 3G, LRU Buffer μεγέθους 500MB και ενεργά χρησιμοποιούμενη μνήμη μεγέθους 1G

- Διάρκεια 100 δευτερολέπτων



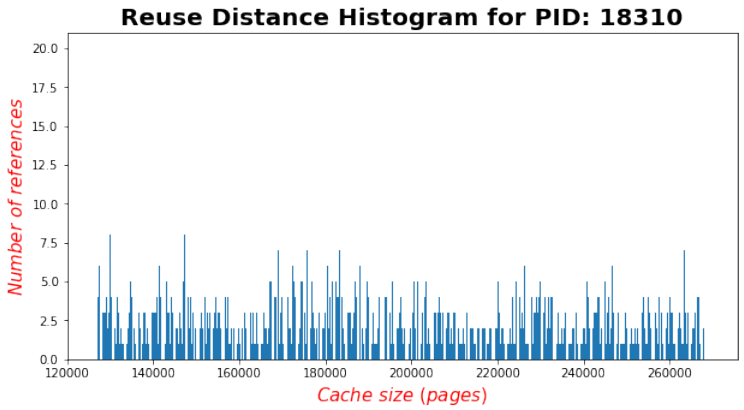
Σχήμα 29: Reuse Distance Histogram, Total Memory 3G, LRU 500M, WSS 1G, 100secs



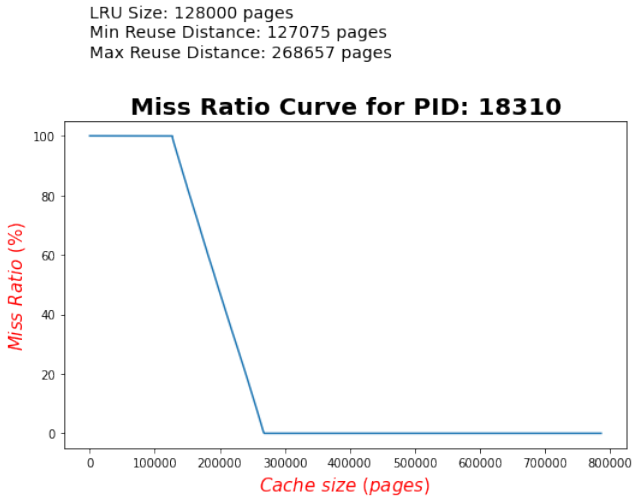
Σχήμα 30: Miss Ratio Curve, Total Memory 3G, LRU 500M, WSS 1G, 100secs

Per- Access Overhead: (282.183usec - 233.955usec) / 233.955usec * 100% = 20%

- Διάρκεια 200 δευτερολέπτων



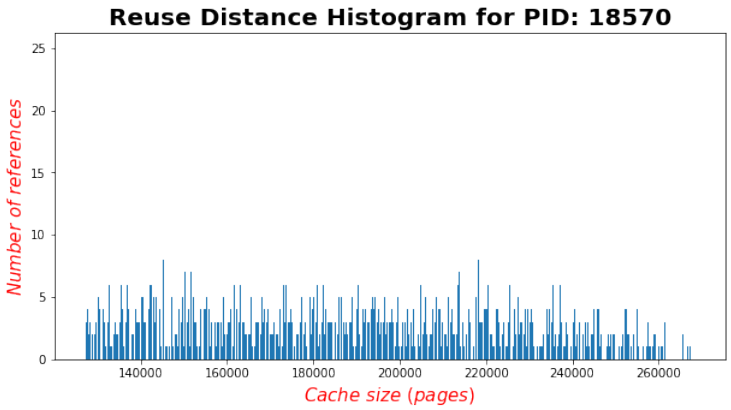
Σχήμα 31: Reuse Distance Histogram, Total Memory 3G, LRU 500M, WSS 1G, 200secs



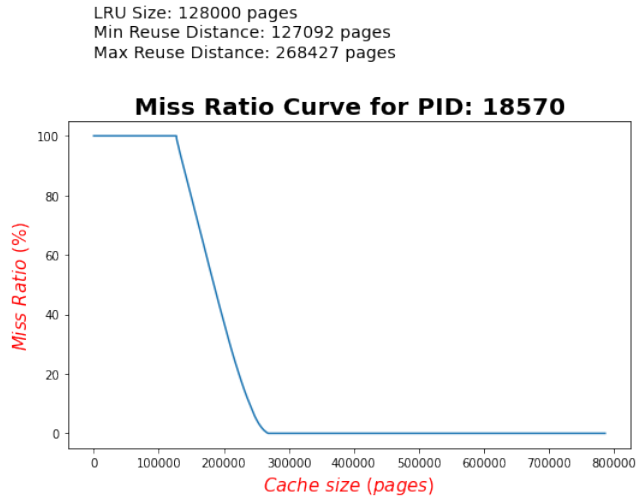
Σχήμα 32: Miss Ratio Curve, Total Memory 3G, LRU 500M, WSS 1G, 200secs

Per- Access Overhead: (336.362usec - 299.406usec)/299.406usec * 100% = 12.3%

- Διάρκεια 1000 δευτερολέπτων



Σχήμα 33: Reuse Distance Histogram, Total Memory 3G, LRU 500M, WSS 1G, 1000secs

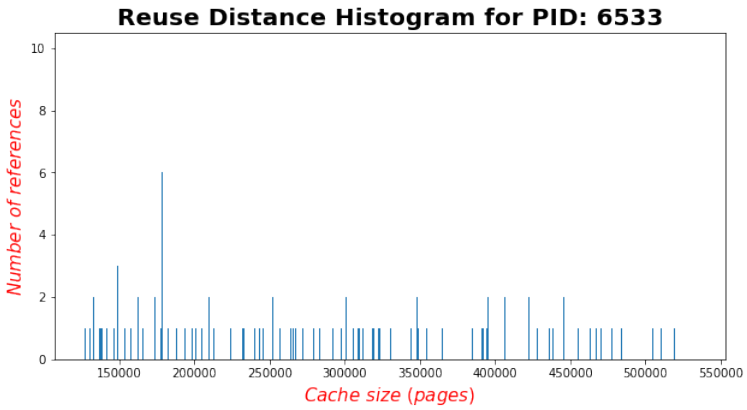


Σχήμα 34: Miss Ratio Curve, Total Memory 3G, LRU 500M, WSS 1G, 1000secs

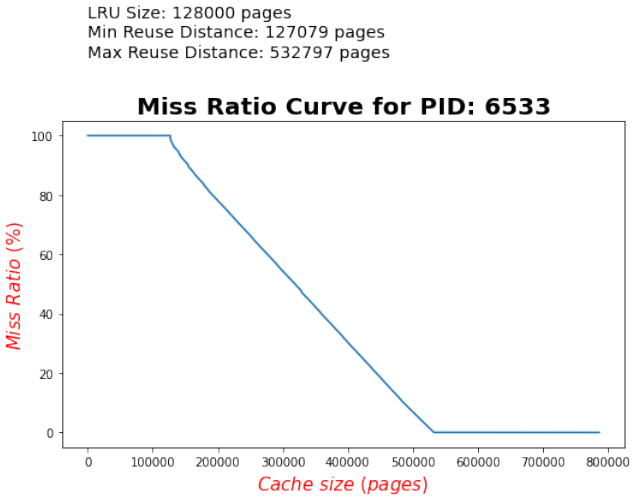
Per- Access overhead: (1231.620usec - 1090.434usec) / 1090.434usec * 100% = 12.9%

Εικονικό μηχάνημα συνολικής μνήμης 3G, LRU Buffer μεγέθους 500MB και ενεργά χρησιμοποιούμενη μνήμη μεγέθους 2G.

- Διάρκεια 100 δευτερολέπτων



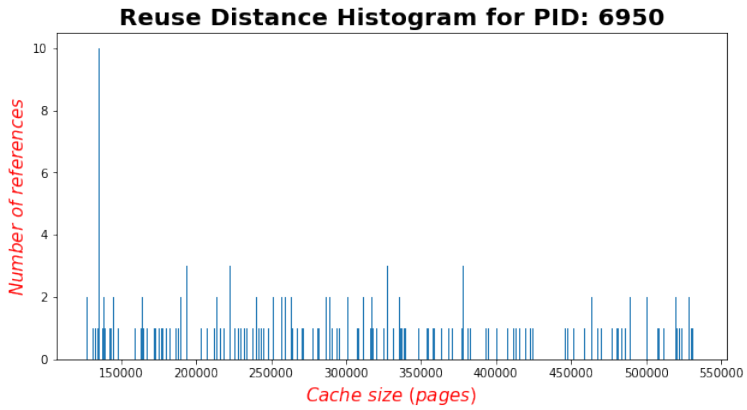
Σχήμα 35: Reuse Distance Histogram, Total Memory 3G, LRU 500M, WSS 2G, 100secs



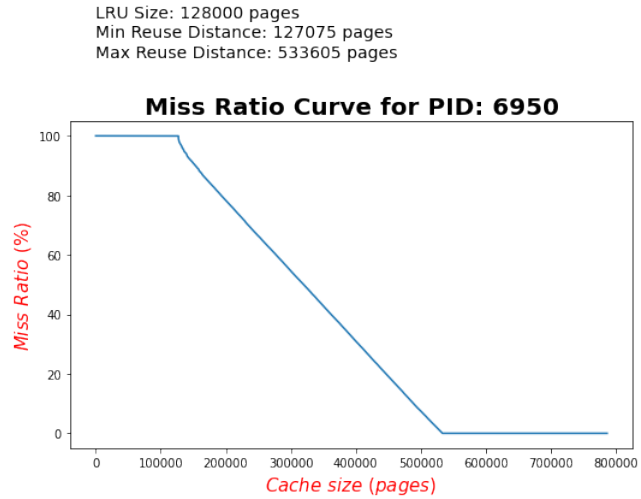
Σχήμα 36: Miss Ratio Curve, Total Memory 3G, LRU 500M, WSS 2G, 100secs

Per- Access overhead: (1145.845 usec - 1131.006 usec / 1131.006 usec)*100% = 1.3%

- Διάρκεια 200 δευτερολέπτων



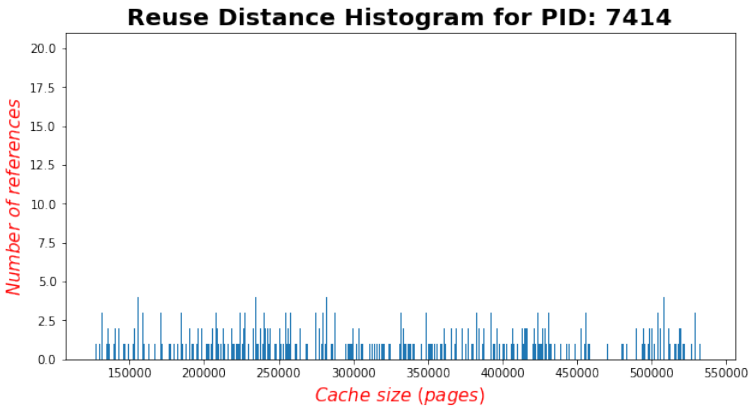
Σχήμα 37: Reuse Distance Histogram, Total Memory 3G, LRU 500M, WSS 2G, 200secs



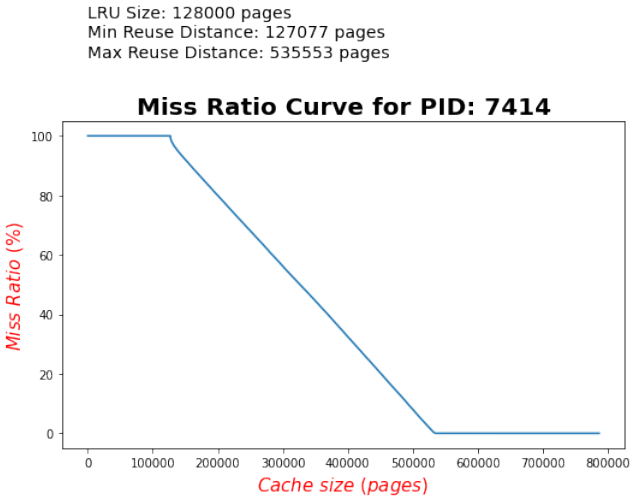
Σχήμα 38: Miss Ratio Curve, Total Memory 3G, LRU 500M, WSS 2G, 200secs

Per- Access overhead: (1216.862 usec - 1056.389 usec) / 1056.389 usec * 100% = 15.2%

- Διάρκεια 1000 δευτερολέπτων



Σχήμα 39: Reuse Distance Histogram, Total Memory 3G, LRU 500M, WSS 2G, 1000secs



Σχήμα 40: Miss Ratio Curve, Total Memory 3G, LRU 500M, WSS 2G, 1000secs

Per- Access overhead: (2570.366 usec - 3504.677 usec) / 3504.677 usec * 100% = -26.65%, δηλαδή ”φαίνεται” σαν να πηγαίνει καλύτερα με ενεργοποίηση του μηχανισμού λήψης μετρικών, γεγονός που φυσικά δεν είναι αληθές και εξηγείται στη συνέχεια.

5.1.5 Παρατηρήσεις

Παρατηρούμε αρχικά ότι λαμβάνουμε κάποιες αποστάσεις επαναχρησιμοποίησης για μεγέθη κοντά αλλά μικρότερα από το μέγεθος του LRU Buffer. Αυτό οφείλεται στο γεγονός ότι κάποιες σελίδες αποτυγχάνουν να μεταφερθούν από το εικονικό μηχάνημα προς την απομακρυσμένη μνήμη. Είναι αναμενόμενες οι μεμονωμένες περιπτώσεις αποτυχίας αν η απομακρυσμένη μνήμη δεν δύναται να δεχθεί προσωρινά αιτήματα (device or resource busy messages). Όταν αποτύχει η μεταφορά κάποιας σελίδας, η σελίδα επανεντάσσεται στον LRUBuffer. Επίσης, η εισαγωγή της γίνεται στην αρχή του. Σε επόμενα λάθη σελίδων, θα γίνει προσπάθεια αφαίρεσης της εκάστοτε επόμενης στη σειρά σελίδας όπως καθορίζεται κατά την LRU πολιτική. Συνήθως, οι αποτυχίες τείνουν να συνεχίζονται για ένα χρονικό διάστημα, έως ότου να αποκατασταθεί η διαθεσιμότητα της μνήμης. Επομένως, κάποιες σελίδες αργότερα θα αφαιρεθούν από το εικονικό μηχάνημα ενώ -θεωρητικά- αυτό δεν ήταν εξαρχής αναμενόμενο. Επαναπροσβάσεις σε αυτό το σύνολο σελίδων που απελευθερώθηκαν από το εικονικό μηχάνημα στη θέση κάποιων άλλων, οδηγούν σε λάθη σελίδων με απόσταση επαναχρησιμοποίησης μικρότερης, αλλά κοντά στο μέγεθος του LRUBuffer.

Επόμενο σημείο παρατήρησης είναι η ακρίβεια της μεθόδου υπολογισμού των αποστάσεων. Στο τρίτο σενάριο στην περίπτωση των διεργασιών, η ελάχιστη απόσταση επαναχρησιμοποίησης σε όλες τις περιπτώσεις είναι ίση με το μέγεθος του LRU Buffer + 1 σελίδα, ενώ η μέγιστη απόσταση επαναχρησιμοποίησης είναι ακριβώς όση το μέγεθος του WSS που ορίσαμε. Στο τέταρτο σενάριο στην περίπτωση των εικονικών μηχανημάτων, για WSS 1G και για 4K μέγεθος σελίδας αναμένουμε μέγιστη απόσταση επαναχρησιμοποίησης κοντά στις 262144 σελίδες, ενώ για WSS 2G αντίστοιχα αναμένουμε μέγιστη απόσταση επαναχρησιμοποίησης κοντά στις 524288 σελίδες. Τα αποτελέσματα που λάβαμε στις δύο περιπτώσεις προκύπτουν εκτός κατά περίπου 6000 σελίδες και 10000 σελίδες αντιστοίχως, που στην περίπτωση των 4KB μεγέθους σελίδας σημαίνει 20MB και 40MB, ένα ποσοστό του 2% του συνολικού WSS. Οπότε εν γένει η μέθοδός μας δίνει ακριβή αποτελέσματα. Ο λόγος που εμφανίζονται περισσότερες σε πλήθος διαφορετικές σελίδες (εμφανείς με μεγαλύτερες αποστάσεις επαναχρησιμοποίησης) στην περίπτωση των 2GB είναι ότι η τυχαιότητα του benchmark συνδυασμένη με το μεγαλύτερο WSS έχει ως αποτέλεσμα αύξηση του χρονικού διαστήματος που απαιτείται για επανεμφάνιση κάποιας σελίδας, οπότε στο διάστημα αυτό υπάρχει χώρος για μεσολάβηση προσβάσεων σε περισσότερες σελίδες εκτός WSS του benchmark.

Τελικό σημείο παρατήρησης είναι το overhead του αλγορίθμου. Στο τρίτο σενάριο βλέπουμε ότι κατά τις πρώτες προσπάθειες με σχετικά μικρό WSS (20MB και 30MB αντιστοίχως), η ταχύτητα εκτέλεσης του πειράματος οδηγεί και σε διακυμάνσεις στο overhead, το οποίο παρατηρήθηκε να λαμβάνει από αρνητική τιμή έως και τιμή της τάξης του 10%. Στην περίπτωση μεγαλύτερου WSS αυξάνει συνολικά και ο χρόνος εκτέλεσης λόγω μεγαλύτερων αλμάτων στη μνήμη, που συνοδεύεται από σχετική σταθεροποίηση του overhead στην τιμή 3% για τα συγκεκριμένα μεγέθη. Τέλος, το παρατηρούμενο overhead του αλγορίθμου έχει μέγιστη τιμή 20%. Εν γένει, όμως, οι χρόνοι εκτέλεσης παρουσιάζονται ασταθείς, είτε με ενεργοποίηση είτε όχι του μηχανισμού λήψης μετρικών. Αυτό οφείλεται σε αποτυχίες του zookeeper instance παροδικά, γεγονός που οδηγεί το σύστημα σε προσωρινή αναμονή και αύξηση του χρόνου εκτέλεσης. Αυτός είναι και ο λόγος εμφάνισης έως και αρνητικού overhead σε ορισμένα tests, ή overhead επιπέδου μόλις 1%.

5.1.6 Τεχνικά θέματα του συστήματος FluidMem

Κατά τη διενέργεια των πειραμάτων, σε ορισμένες περιπτώσεις που σχετίζονται κυρίως με μεγάλη δραστηριότητα μεταφοράς σελίδων προς την απομακρυσμένη μνήμη και παράλληλη διατήρηση σχετικά μικρού μεγέθους τοπικής μνήμης (LRU Buffer) έχουν παρατηρηθεί τα εξής τεχνικά θέματα:

Αδυναμία εξόδου από την ioctl(UFFDIO_REMAP)

Στην κλήση της ioctl(UFFDIO_REMAP) στη διαδρομή κατά το eviction σελίδων έχει παρατηρηθεί αδυναμία εξόδου. Με επιπλέον ενεργοποίηση της βελτιστοποίησης –enable-threadedwrite παρατηρείται επίσης αδυναμία εξόδου από την munmap συνάρτηση στον ίδιο χρόνο. Κλήση της strace δείχνει unfinished futexes σε αυτές τις περιπτώσεις.

SIGBUS με μικρό μέγεθος LRU

Επίσης συναντιούνται περιπτώσεις κατά τις οποίες σε τυχαίους χρόνους και μόνο όταν το μέγεθος του LRU Buffer είναι σχετικά μικρό (μερικές εκατοντάδες σελίδες) η διεργασία της οποίας η μνήμη ελέγχεται από το FluidMem λαμβάνει SIGBUS error. Αυτό οφείλεται σε μη - υλοποιημένη διόρθωση του UFFDIO_REMAP patch [36], ώστε να διαχειρίζεται την περίπτωση παράλληλης εμφάνισης λάθους σελίδας και κλήσης της ioctl(UFFDIO_REMAP) στη σελίδα αυτή. Πρόκειται για περίπτωση που εμφανίζεται σε μικρά μεγέθη LRU Buffer διότι σε αντίθετη περίπτωση πολύ σπάνια επιλέγεται να γίνει remapped out σελίδα στην οποία μόλις παρατηρείται pagefault.

Αποτυχίες του Zookeeper

Παρατηρείται αύξηση σε χρόνο εκτέλεσης διεργασιών εξαιτίας παροδικών αποτυχιών του zookeeper instance που επανέρχεται έπειτα από λίγο. Σπάνια, έχει επίσης παρατηρηθεί αποτυχία της monitor διεργασίας να επικοινωνήσει με το zookeeper instance λόγω μηνύματος Broken pipe.

Αποτυχίες του RAMCloud

Έχει διαπιστωθεί η εμφάνιση μηνυμάτων Device or resource busy για την απομακρυσμένη μνήμη. Έτσι προσωρινά παγώνει η μεταφορά σελίδων από το τοπικό μηχάνημα στη μνήμη αυτή.

Κεφάλαιο 6

Συμπεράσματα και Μελλοντική Εργασία

6.1 Σύνοψη συμπερασμάτων αξιολόγησης

Συνοψίζοντας, η ένταξη του αλγορίθμου του Olken στο μηχανισμό σελιδοποίησης του FluidMem από την μια δεν εισάγει απαγορευτικό overhead, ενώ από την άλλη παρέχει ιδιαίτερη ακρίβεια υπολογισμού του μεγέθους της ενεργά χρησιμοποιούμενης μνήμης των εικονικών μηχανημάτων, για μεγέθη μεγαλύτερα της χωρητικότητας του LRU Buffer.

Έτσι, ως πρώτη αξιοποίηση της γνώσης, η τοπική μνήμη μέσω του LRU buffer μπορεί να διατεθεί καταλληλότερα με αύξηση της χωρητικότητας του Buffer όταν παρατηρείται ενεργά χρησιμοποιούμενη μνήμη που υπερβαίνει το μέγεθός του. Επίσης, όμως, οι δυνατότητες επεκτείνονται με ευκαιρίες για αυξομείωση της απομακρυσμένης hot-added μνήμης που διατίθεται ανά εικονικό μηχάνημα και άρα αποδοτικότερη διαχείριση στο σύνολο των πόρων μνήμης του συμπλέγματος.

6.2 Μελλοντικές Επεκτάσεις

Μέσω του μηχανισμού hotplug, η υπάρχουσα υλοποίηση του FluidMem είναι ικανή να παρέχει με διαφάνεια μνήμη που ανήκει σε απομακρυσμένους κόμβους στα φιλοξενούμενα εικονικά μηχανήματα. Οπότε, με γνώση του μεγέθους συνόλου εργασίας, είναι εφικτή η αύξηση της μνήμης των εικονικών μηχανημάτων σε περιπτώσεις που θεωρείται επιθυμητό.

Η υποστήριξη της αύξησης και της ελάττωσης της μνήμης των μηχανημάτων μέσω hotplug και hot-unplug δυναμικά θα μπορούσε να ενταχθεί στο FluidMem ως μελλοντική εργασία. Σαν πρώτο βήμα, θα πρέπει να επεκταθεί η hot-unplug διεπαφή ώστε μνήμη του FluidMem να είναι εφικτό να αφαιρείται από ένα εικονικό μηχάνημα. Σαν επόμενο βήμα, η monitor διεργασία θα πρέπει να είναι σε θέση να εκτελεί αιτήματα αύξησης και μείωσης της μνήμης των εικονικών μηχανημάτων ανάλογα με την εκάστοτε εικόνα του συνόλου εργασίας τους.

Τέλος, θα μπορούσε να μειωθεί το overhead του αλγορίθμου αν γινόταν αξιοποίηση της μεθόδου SHARDS, για επιλογή ενός δείγματος σελίδων για τις οποίες θα υπολογίζουμε τις αποστάσεις επαναχρησιμοποίησης.

Βιβλιογραφία

[1] L. Liu, W. Cao, S. Sahin, Q. Zhang, J. Bae, and Y. Wu, "Memory Disaggregation: Research Problems and Opportunities," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 1664–1673.

[2] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient Memory Disaggregation with Infiniswap," en, 2017, pp. 649–667, isbn: 978-1-931971-37-9. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu (visited on 09/01/2020).

[3] B. Caldwell, "Fluidmem: Open Source Full Memory Disaggregation," en, p. 74,

[4] B. Caldwell, S. Goodarzy, S. Ha, R. Han, E. Keller, E. Rozner, and Y. Im, "FluidMem: Full, Flexible, and Fast Memory Disaggregation for the Cloud," en, p. 13,

[5] Geeta and S. Prakash, "Role of Virtualization Techniques in Cloud Computing Environment," in. Aug. 2019, pp. 439–450. doi: 10.1007/978-981-13-0344-9_37.

[6] M. Anisetti, V. Bellandi, A. Colombo, M. Cremonini, E. Damiani, F. Frati, J. Hounsou, and D. Rebeccani, "Learning Computer Networking on Open Paravirtual Laboratories," *Education, IEEE Transactions on*, vol. 50, pp. 302–311, Dec. 2007. doi: 10.1109/TE.2007.904584.

[7] S. Reth, "Tracking Working Set Sizes of Virtual Machines Using Miss Ratio Curves," en, p. 75,

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," en, p. 14,

[9] P. Denning, "Working Sets Past and Present," en, *IIEEE Trans. Software Eng.*, vol. SE-6, no. 1, pp. 64–84, Jan. 1980, issn: 0098-5589. doi: 10.1109/TSE.1980.230464. [Online]. Available: http://ieeexplore.ieee.org/document/1702696/ (visited on 09/02/2020).

[10] D. Byrne, "A Survey of Miss-Ratio Curve Construction Techniques," *arXiv:1804.01972 [cs]*, Apr. 2018, arXiv: 1804.01972. [Online]. Available: http://arxiv.org/abs/1804.01972 (visited on 08/23/2020).

[11] C. A. Waldspurger, T. Saemundsson, I. Ahmad, and N. Park, "Miniature Cache Simulations for Modeling and Optimization," en, vol. 43, no. 1, p. 8, 2018.

[12] D. Byrne, "mPart: Miss Ratio Curve Guided Partitioning in Key-Value Stores," en, p. 62,

[13] W. Zhao, X. Jin, Z. Wang, X. Wang, Y. Luo, and X. Li, "Low Cost Working Set Size Tracking," en, p. 6,

[14] C. Waldspurger, N. Park, A. Garthwaite, and I. Ahmad, "Efficient MRC Construction with SHARDS," Feb. 2015.

[15] X. Hu, X. Wang, L. Zhou, Y. Luo, C. Ding, and Z. Wang, "Kinetic Modeling of Data Eviction in Cache," en, p. 15,

[16] C. Fang, S. Carr, S. Önder, and Z. Wang, "Reuse-distance-based miss-rate prediction on a per instruction basis," in *Proceedings of the 2004 workshop on Memory system performance*, ser. MSP '04, New York, NY, USA: Association for Computing Machinery, Jun. 2004, pp. 60–68, isbn: 978-1-58113-941-9. doi: 10.1145/1065895.1065906. [Online]. Available: https://doi.org/10.1145/1065895.1065906 (visited on 09/01/2020).

[17] C. Ding and Y. Zhong, “Predicting Whole-Program Locality through Reuse Distance Analysis,” en, p. 13,

[18] F. Olken, *Efficient methods for calculating the success function of fixed- space replacement policies.* 1981.

[19] R. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, “Evaluation techniques for storage hierarchies,” en, *IBM Syst. J.*, vol. 9, no. 2, pp. 78–117, 1970, issn: 0018-8670. doi: 10.1147/sj.92.0078. [Online]. Available: http://ieeexplore.ieee.org/document/5388318/ (visited on 08/26/2020).

[20] *(PDF) Instruction based memory distance analysis and its application to optimization*, en. [Online]. Available: https://www.researchgate.net/publication/4178702_Instruction_based_memory_distance_analysis_and_its_application_to_optimization (visited on 09/01/2020).

[21] B. T. Bennett and V. J. Kruskal, “LRU Stack Processing,” en, *IBM J. Res. & Dev.*, vol. 19, no. 4, pp. 353–357, Jul. 1975, issn: 0018-8646, 0018-8646. doi: 10.1147/rd.194.0353. [Online]. Available: http://ieeexplore.ieee.org/document/5391172/ (visited on 09/27/2020).

[22] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “Geiger: Monitoring the Buffer Cache in a Virtual Machine Environment,” en, p. 11,

[23] *(19) (PDF) Dynamic tracking of page miss ratio curve for memory management*, en. [Online]. Available: https://www.researchgate.net/publication/220938765_Dynamic_tracking_of_page_miss_ratio_curve_for_memory_management (visited on 09/03/2020).

[24] B. Caldwell, Y. Im, S. Ha, R. Han, and E. Keller, “FluidMem: Memory as a Service for the Datacenter,” en, *arXiv:1707.07780 [cs]*, Jul. 2017, arXiv: 1707.07780. [Online]. Available: http://arxiv.org/abs/1707.07780 (visited on 08/24/2020).

[25] D. E. Comer and J. Griffioen, “A New Design for Distributed Systems: The Remote Memory Model,” en, p. 17,

[26] A. Samih, R. Wang, C. Maciocco, C. Tai, R. Duan, J. Duan, and Y. Solihin, “Evaluating Dynamics and Bottlenecks of Memory Collaboration in Cluster Systems,” *Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012*, May 2012. doi: 10.1109/CCGrid.2012.59.

[27] [Online]. Available: https://www.kernel.org/doc/Documentation/vm/userfaultfd.txt (visited on 08/24/2020).

[28] *blakecaldwell/userfault-kernel*, en. [Online]. Available: https://github.com/blakecaldwell/userfault-kernel (visited on 08/24/2020).

[29] *RumblePhd.pdf*. [Online]. Available: https://web.stanford.edu/~ouster/cgi-bin/papers/RumblePhd.pdf (visited on 08/24/2020).

[30] J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S. J. Park, H. Qin, M. Rosenblum, S. Rumble, R. Stutsman, and S. Yang, “The RAMCloud Storage System,” en, *ACM Trans. Comput. Syst.*, vol. 33, no. 3, pp. 1–55, Sep. 2015, issn: 0734-2071, 1557-7333. doi: 10.1145/2806887. [Online]. Available: https://dl.acm.org/doi/10.1145/2806887 (visited on 08/24/2020).

[31] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, “ZooKeeper: Wait-free coordination for Internet-scale systems,” en, p. 14,

[32] *logcabin/logcabin*, original-date: 2012-06-25T21:09:45Z, Aug. 2020. [Online]. Available: https://github.com/logcabin/logcabin (visited on 08/30/2020).

[33] *Splay Trees*. [Online]. Available: http://www.cs.cornell.edu/courses/cs3110/2011sp/Recitations/rec25-splay/splay.htm (visited on 10/15/2020).

[34] E. K. Lee and C. U. Martel, “When to use splay trees,” en, *Software: Practice and Experience*, vol. 37, no. 15, pp. 1559–1575, 2007, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.813, issn: 1097-024X. doi: 10.1002/spe.813. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.813 (visited on 10/20/2020).

[35] *jisooy / pmbench — Bitbucket*. [Online]. Available: https://bitbucket.org/jisooy/pmbench/src/master/ (visited on 09/20/2020).

[36] *RFC: userfaultfd v3 [LWN.net]*. [Online]. Available: https://lwn.net/Articles/635608/ (visited on 10/03/2020).