Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

τομεας τεχνολογιας πληροφορικης και υπολογιστων

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

Machine Learning-Driven Workload Clustering and Placement for Heterogeneous DRAM/NVM Memory Systems

Μελέτη και υλοποίηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ιωάννη Ρόκομου

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Machine Learning-Driven Workload Clustering and Placement for Heterogeneous DRAM/NVM Memory Systems

Μελέτη και υλοποίηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ιωάννη Ρόκομου

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 24η Ιουνίου, 2025.

.....	.....	.....
Δημήτριος Σούντρης	Σωτήριος Ξύδης	Γεώργιος Ζερβάκης
Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.	Καθηγητής Π.Π.

Αθήνα, Ιούνιος 2025

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

Περίληψη

Τα σύγχρονα υπολογιστικά συστήματα αντιμετωπίζουν ολοένα και περισσότερο το πρόβλημα του περιορισμού της μνήμης, όπου οι ταχύτητες πρόσβασης στη μνήμη δεν συμβαδίζουν με τις βελτιώσεις στους επεξεργαστές. Αυτός ο περιορισμός επηρεάζει αρνητικά την απόδοση των εφαρμογών που απαιτούν έντονη χρήση μνήμης, καθιστώντας απαραίτητες καινοτόμες λύσεις για την αποδοτική διαχείριση των διαθέσιμων πόρων μνήμης. Μία πολλά υποσχόμενη προσέγγιση είναι η χρήση ετερογενών αρχιτεκτονικών μνήμης που συνδυάζουν την παραδοσιακή DRAM με νέες τεχνολογίες μη πτητικής μνήμης, όπως η Intel Optane. Ωστόσο, η αποδοτική αξιοποίηση τέτοιων συστημάτων απαιτεί έξυπνες στρατηγικές ταξινόμησης φορτίων εργασίας και κατανομής μνήμης.

Η παρούσα μελέτη διερευνά τη χρήση τεχνικών μηχανικής μάθησης για την ταξινόμηση φορτίων εργασίας με βάση τα χαρακτηριστικά τους σε επίπεδο μνήμης και επεξεργαστή. Δημιουργείται ένα σύνολο δεδομένων μέσω της ανάλυσης διάφορων benchmarks, εξάγοντας κρίσιμες μετρικές απόδοσης που σχετίζονται με το εύρος ζώνης της μνήμης και τη χρήση της CPU. Στη συνέχεια, εκπαιδεύονται διάφορα μοντέλα επιβλεπόμενης μάθησης, όπως το Random Forest, το K-Nearest Neighbors και το Naive Bayes, με σκοπό την κατηγοριοποίηση των φορτίων εργασίας σε προκαθορισμένες κλάσεις. Επιπλέον, εφαρμόζονται τεχνικές επιλογής χαρακτηριστικών και μείωσης διαστάσεων, όπως η Ανάλυση Κύριων Συνιστωσών (PCA), για τη βελτίωση της απόδοσης και της ερμηνευσιμότητας των μοντέλων.

Λέξεις Κλειδιά — Ετερογενής Μνήμη, Intel Optane, Μηχανική Μάθηση, Ταξινόμηση Φορτίων Εργασίας, Ανάλυση Κύριων Συνιστωσών, Κατανομή Πόρων

Abstract

Modern computing systems increasingly face memory bottlenecks, where memory access speeds fail to keep pace with processor advancements. This limitation hinders the performance of memory-intensive applications, necessitating innovative solutions to efficiently manage memory resources. One promising approach is the integration of heterogeneous memory architectures that combine traditional DRAM with emerging non-volatile memory technologies such as Intel Optane. However, leveraging such architectures effectively requires intelligent workload classification and memory allocation strategies.

This study explores the application of machine learning techniques to classify workloads based on their memory and computational characteristics. A dataset is constructed by profiling various benchmarks and extracting key performance metrics related to memory bandwidth and CPU utilization. Several supervised learning models, including Random Forest, K-Nearest Neighbors, and Naive Bayes, are trained on this dataset to categorize workloads into predefined classes. Feature selection and dimensionality reduction techniques, such as Principal Component Analysis (PCA), are employed to enhance model performance and interpretability.

Keywords — Heterogeneous Memory, Intel Optane, Machine Learning, Workload Classification, Principal Component Analysis, Resource Allocation

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον καθηγητή κ. Δημήτριο Σούντρη, ο οποίος μου έδωσε την ευκαιρία να εργαστώ πάνω σε αυτή τη διπλωματική εργασία στο εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων. Η εμπιστοσύνη που μου έδειξε ήταν καθοριστική για την έναρξη αυτής της προσπάθειας και για την εξέλιξή μου στον τομέα. Ευχαριστώ επίσης τον υποψήφιο διδάκτορα Μανώλη Κατσαραγκάκη για την καθοδήγηση και τη συνεχιζόμενη υποστήριξή του κατά τη διάρκεια της εκπόνησης της εργασίας. Η βοήθειά του υπήρξε πολύτιμη και η εμπειρία του στο αντικείμενο με βοήθησε να αναπτύξω τις ιδέες μου και να ολοκληρώσω την εργασία με επιτυχία.

Ευχαριστώ την οικογένειά μου για την αγάπη και τη στήριξή τους καθ' όλη τη διάρκεια της πορείας μου. Τέλος, ευχαριστώ τους φίλους μου για την ηθική στήριξη η οποία με βοήθησε να παραμείνω επικεντρωμένος και να ξεπεράσω τις προκλήσεις της δουλειάς μου.

Contents

Figure List

Table List

Εκτεταμένη Ελληνική Περίληψη

Εισαγωγή

Τα σύγχρονα υπολογιστικά συστήματα αντιμετωπίζουν σημαντικές προκλήσεις απόδοσης λόγω του περιορισμού της μνήμης (memory bottleneck), ένα κρίσιμο ζήτημα όπου η ταχύτητα πρόσβασης στα δεδομένα από τη μνήμη δεν μπορεί να συμβαδίσει με την υπολογιστική ισχύ των επεξεργαστών. Καθώς οι επεξεργαστές συνεχίζουν να εξελίσσονται, γίνονται ταχύτεροι και πιο αποδοτικοί, όμως οι περιορισμοί στην πρόσβαση στη μνήμη αποτελούν σημαντικό εμπόδιο για την επίτευξη βέλτιστης απόδοσης. Αυτός ο περιορισμός προκύπτει επειδή η DRAM, η κύρια πτητική μνήμη που χρησιμοποιείται στα περισσότερα συστήματα, δεν έχει αυξήσει την ταχύτητά της με τον ίδιο ρυθμό όπως οι επεξεργαστές. Ως αποτέλεσμα, εφαρμογές που βασίζονται σε συχνή πρόσβαση στη μνήμη υφίστανται καθυστερήσεις και μειωμένη αποδοτικότητα.

Μία πιθανή λύση για την αντιμετώπιση του προβλήματος αυτού είναι η αύξηση της συνολικής χωρητικότητας της μνήμης ενός συστήματος με τη χρήση πιο οικονομικών τεχνολογιών μνήμης, όπως η Intel Optane. Η Optane παρέχει έναν αποδοτικό τρόπο επέκτασης της διαθέσιμης μνήμης, ωστόσο τα χαρακτηριστικά απόδοσής της διαφέρουν από την παραδοσιακή DRAM, ειδικά όσον αφορά την καθυστέρηση (latency) και το εύρος ζώνης (bandwidth). Επομένως, απαιτείται μια αποδοτική στρατηγική διαχείρισης που να βελτιστοποιεί τη χρήση τόσο της DRAM όσο και της Optane στην ίδια αρχιτεκτονική. Ο σωστός χαρακτηρισμός των εργασιών και η έξυπνη κατανομή δεδομένων μπορούν να βοηθήσουν στην επίτευξη υψηλής απόδοσης εκμεταλλευόμενοι τη συνδυαστική χρήση των δύο τύπων μνήμης.

Για την αποδοτική αξιοποίηση ενός ετερογενούς συστήματος μνήμης, είναι απαραίτητο να κατανοηθεί ο τρόπος με τον οποίο διαφορετικές εφαρμογές αλληλεπιδρούν με τους διαθέσιμους πόρους μνήμης. Οι εφαρμογές παρουσιάζουν ποικίλες συμπεριφορές, με κάποιες να είναι περισσότερο μνημονικά εντατικές, ενώ άλλες να βασίζονται κυρίως στην υπολογιστική ισχύ. Η δυνατότητα ταξινόμησης των εργασιών βάσει των προτύπων πρόσβασης στη μνήμη επιτρέπει καλύτερες στρατηγικές κατανομής, διασφαλίζοντας ότι οι κρίσιμες για την απόδοση εφαρμογές λαμβάνουν τους ταχύτερους διαθέσιμους πόρους μνήμης.

Στην παρούσα μελέτη, χρησιμοποιούνται τεχνικές μηχανικής μάθησης για την ταξινόμηση των εφαρμογών με βάση τη συμπεριφορά τους όσον αφορά τη χρήση της μνήμης και της υπολογιστικής ισχύος. Μέσω του χαρακτηρισμού διαφόρων εργασιών και της εξαγωγής κρίσιμων μετρικών απόδοσης, δημιουργείται ένα σύνολο δεδομένων το οποίο χρησιμοποιείται για την εκπαίδευση εποπτευόμενων μοντέλων μηχανικής μάθησης. Ο στόχος είναι η ανάπτυξη μιας μεθοδολογίας που μπορεί να ταξινομεί με ακρίβεια τις εφαρμογές και να παρέχει χρήσιμες πληροφορίες σχετικά με τις απαιτήσεις τους στη μνήμη. Αυτή η ταξινόμηση μπορεί να καθοδηγήσει στρατηγικές διαχείρισης μνήμης, βελτιώνοντας τη συνολική αποδοτικότητα ετερογενών αρχιτεκτονικών μνήμης.

Η προτεινόμενη προσέγγιση περιλαμβάνει τον χαρακτηρισμό εργασιών που εκτελούνται τόσο σε DRAM όσο και σε Optane, την εξαγωγή σχετικών χαρακτηριστικών απόδοσης και τη χρήση μοντέλων μηχανικής μάθησης για την ταξινόμηση των εφαρμογών σε προκαθορισμένες κατηγορίες. Επιπλέον, εφαρμόζονται τεχνικές επιλογής χαρακτηριστικών και μείωσης διαστάσεων για τη βελτίωση της ακρίβειας της ταξινόμησης και τη μείωση της υπολογιστικής πολυπλοκότητας. Τα εκπαιδευμένα μοντέλα αξιολογούνται σε πολλαπλά σενάρια για να εξεταστεί η αποτελεσματικότητά τους στη διάκριση διαφορετικών τύπων εφαρμογών.

Σχετική Βιβλιογραφία

Η εμφάνιση των υβριδικών συστημάτων μνήμης που συνδυάζουν την υψηλής ταχύτητας DRAM με τις μη πτητικές μνήμες υψηλής χωρητικότητας, όπως η Intel Optane Persistent Memory (PMEM), έχει προκαλέσει σημαντικό ενδιαφέρον για τις τεχνικές δυναμικής διαχείρισης μνήμης. Ένα μεγάλο σύνολο ερευνών έχει εξετάσει τη δυναμική τοποθέτηση των δεδομένων σε τέτοια συστήματα με στόχο τη βελτιστοποίηση της απόδοσης και της αξιοποίησης των πόρων.

Οι περισσότερες υπάρχουσες προσεγγίσεις στη διαχείριση υβριδικής μνήμης επικεντρώνονται στη μετακίνηση και τοποθέτηση δεδομένων, είτε στο **επίπεδο αντικειμένων** είτε στο **επίπεδο σελίδας**. Αυτές οι μέθοδοι εξαρτώνται σε μεγάλο βαθμό από τον προφίλ εκτέλεσης κατά το χρόνο εκτέλεσης, για να παρακολουθήσουν τη συμπεριφορά της πρόσβασης στη μνήμη και χρησιμοποιούν τις συλλεγόμενες πληροφορίες για να καθοδηγήσουν τις αποφάσεις σχετικά με το πού πρέπει να τοποθετούνται τα δεδομένα - είτε στην DRAM για γρήγορη πρόσβαση, είτε στην PMEM για μεγαλύτερη χωρητικότητα. Αν και είναι αποτελεσματικές στη βελτίωση της απόδοσης του συστήματος, αυτές οι στρατηγικές συχνά συνεπάγονται υψηλό κόστος παρακολούθησης και μπορεί να απαιτούν συχνή παρακολούθηση και μετανάστευση των δεδομένων, κάτι που μπορεί να αναιρέσει κάποια από τα οφέλη της απόδοσης της DRAM.

Στην κατηγορία της **τοποθέτησης αντικειμένων**, ο διαχειριστής μνήμης παρακολουθεί τη συμπεριφορά των μεμονωμένων δομών δεδομένων ή των κατανομών μνήμης για να προσδιορίσει την καλύτερη τοποθέτηση εντός υβριδικών συστημάτων μνήμης. Μία προσέγγιση εισάγει έναν διαφανή και αποδοτικό μηχανισμό τοποθέτησης αντικειμένων σε διαμορφώσεις DRAM-NVM, με στόχο τη μείωση της κυκλοφορίας εγγραφών στη μη πτητική μνήμη και τη βελτίωση της απόδοσης χωρίς σημαντικές αλλαγές στις διαχειριζόμενες ρουτίνες [1]. Μία άλλη μέθοδος προτείνει μία πολιτική κατανομής που καθοδηγείται από την απόδοση, η οποία επιλέγει δυναμικά τους τύπους μνήμης με βάση την αναμενόμενη επίδραση στην απόδοση [2]. Μια στρατηγική σε επίπεδο αντικειμένου, που βασίζεται στην αντιστοίχιση αντικειμένων κατά τη διάρκεια του χρόνου εκτέλεσης και την τοποθέτηση με γνώση του εύρους ζώνης, έχει επίσης αποδειχθεί ότι ενισχύει σημαντικά την απόδοση σε υβριδικές διαμορφώσεις DRAM-PMEM [3]. Επιπλέον, μελέτες που χρησιμοποιούν εργαλεία προφίλ σε πραγματικό υλικό αποδεικνύουν ότι η βελτιστοποίηση της τοποθέτησης δεδομένων σε περιβάλλοντα ετερογενών μνημών μπορεί να προσφέρει σημαντικά κέρδη τόσο στην απόδοση όσο και στην ενεργειακή αποδοτικότητα [4].

Αντίθετα, οι προσεγγίσεις **τοποθέτησης σελίδων** λειτουργούν στο επίπεδο του λειτουργικού συστήματος και βασίζονται σε παρακολούθηση σελίδων για την ανίχνευση προτύπων πρόσβασης. Μία προσέγγιση προτείνει ένα νέο, ολιστικό πλαίσιο για υβριδικές αρχιτεκτονικές μνήμης, συνδυάζοντας δυναμική τοποθέτηση δεδομένων με μόνιμους σωρούς και περιοδική μόνιμη αποθήκευση, προκειμένου να βελτιστοποιήσει την απόδοση σε εφαρμογές υψηλών επιδόσεων. Αυτή η τεχνική μειώνει τους επαναλαμβανόμενους υπολογισμούς και ενισχύει τη συνολική αποδοτικότητα του συστήματος [5]. Μια άλλη μελέτη παρουσιάζει έναν δυναμικό αλγόριθμο τοποθέτησης σελίδων για υβριδικά συστήματα DRAM-DCPMM, βελτιστοποιώντας τις πολιτικές μνήμης για να βελτιώσει τόσο την απόδοση όσο και την ενεργειακή αποδοτικότητα [6]. Επιπλέον, έχει μελετηθεί η επίδραση της τοποθέτησης των σελίδων πίνακα σε συστήματα κλιμακωτής μνήμης, με μια προτεινόμενη μέθοδο για την αποδοτική διαχείριση των σελίδων πίνακα μετακινώντας τις δυναμικά μεταξύ DRAM και NVMM [7]. Μία περαιτέρω μελέτη εισάγει ένα σχήμα διαχείρισης σελίδων για συστήματα πολλαπλών επιπέδων μνήμης που επεκτείνει την υποστήριξη NUMA, βελτιστοποιώντας την τοποθέτηση σελίδων με βάση τόσο την τοπικότητα πρόσβασης όσο και το επίπεδο μνήμης [8].

Παρά την ποικιλία και ωριμότητα αυτών των τεχνικών, μοιράζονται ένα κοινό σημείο: τη βελτιστοποίηση της τοποθέτησης των *δεδομένων*. Αντίθετα, αυτή η εργασία ασχολείται με μια διαφορετική και λιγότερο εξερευνημένη πτυχή της διαχείρισης υβριδικής μνήμης - την **τοποθέτηση διεργασιών ή εργασιών**.

Με βάση τις διαθέσιμες πληροφορίες, καμία προηγούμενη εργασία δεν έχει προτείνει ένα πλαίσιο τοποθέτησης που να εκμεταλλεύεται τη ταξινόμηση με βάση τη μηχανική μάθηση για να καθοδηγήσει την τοποθέτηση των εργασιών σε διαφορετικούς τύπους μνήμης σε ένα υβριδικό σύστημα. Στην παρούσα προσέγγισή, οι εργασίες αναλύονται μία φορά κατά τη διάρκεια μιας ελεγχόμενης φάσης εκπαίδευσης, και στη συνέχεια χρησιμοποιείται ένας ταξινομητής κατά τη διάρκεια εκτέλεσης για να προβλέψει τα χαρακτηριστικά πρόσβασης στη μνήμη τους (π.χ., έντονες αναγνώσεις, έντονες εγγραφές). Με βάση αυτές

τις προβλέψεις, οι εργασίες τοποθετούνται στην DRAM ή την PMEM, χωρίς να απαιτείται συνεχής παρακολούθηση ή μετανάστευση δεδομένων.

Αυτή η στρατηγική μειώνει σημαντικά το κόστος εκτέλεσης, καθώς δεν είναι απαραίτητο να παρακολουθούνται συνεχώς οι προσβάσεις στη μνήμη ή να μετακινούνται σελίδες και αντικείμενα κατά τη διάρκεια της εκτέλεσης. Επιπλέον, η μεθοδολογία παρέχει μια ελαφριά και κλιμακούμενη προσέγγιση για συστήματα που εκτελούν πολλές σύντομες εργασίες, όπου το κόστος της παρακολούθησης και μετανάστευσης δεδομένων μπορεί να είναι απαγορευτικό.

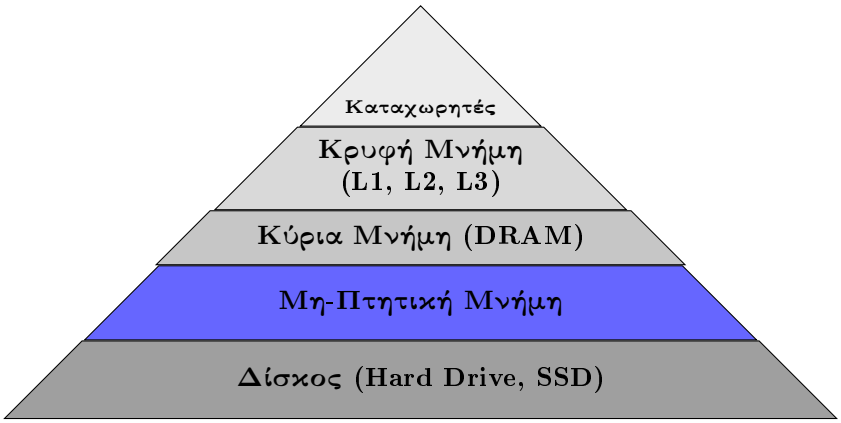
Συνοψίζοντας, ενώ έχει γίνει σημαντική έρευνα στη διαχείριση υβριδικής μνήμης μέσω της τοποθέτησης αντικειμένων και σελίδων, αυτή η εργασία εισάγει μια νέα μέθοδο χαμηλού κόστους που επικεντρώνεται στην τοποθέτηση εργασιών με βάση την ταξινόμηση μέσω μηχανικής μάθησης - προσφέροντας τόσο κλιμακωσιμότητα όσο και βελτιωμένη αξιοποίηση πόρων για συστήματα πολλαπλών εργασιών.

Θεωρητικό Υπόβαθρο

Μη-Πτητική Μνήμη

Η μη-πτητική μνήμη (PMEM) συνδυάζει χαρακτηριστικά της DRAM και των αποθηκευτικών μονάδων (SSDs), προσφέροντας υψηλή ταχύτητα με διατήρηση δεδομένων μετά από απώλεια ρεύματος. Καλύπτει το κενό μεταξύ της DRAM και των SSDs, επιτρέποντας ταχύτερη πρόσβαση και χαμηλότερη κατανάλωση ενέργειας. Τεχνολογίες όπως οι ReRAM, PCM, STT-RAM και 3D XPoint έχουν αναπτυχθεί για την υλοποίησή της.

Η NVM μπορεί να χρησιμοποιηθεί είτε ως αποθηκευτικός χώρος είτε ως εναλλακτική της DRAM. Ως αποθήκευση, προσφέρει χαμηλή καθυστέρηση και χαμηλή κατανάλωση, αλλά έχει περιορισμένη αντοχή σε κύκλους εγγραφής. Ως κύρια μνήμη, παρέχει υψηλότερη πυκνότητα και σχεδόν μηδενική στατική κατανάλωση ισχύος, αλλά μειωμένο εύρος ζώνης και μεγαλύτερο χρόνο πρόσβασης σε σχέση με τη DRAM.



Εικόνα 1: Ιεραρχία Μνήμης

Intel's Optane DIMM

Η Intel Optane DC Persistent Memory (PMEM) είναι η πρώτη εμπορικά διαθέσιμη κλιμακούμενη NVDIMM. Σε αντίθεση με τα SSDs, συνδέεται απευθείας στον δίαυλο μνήμης της CPU, μειώνοντας την καθυστέρηση και αυξάνοντας το εύρος ζώνης. Είναι διαθέσιμη σε χωρητικότητες 128GB, 256GB και 512GB και χρησιμοποιείται σε επεξεργαστές Intel Xeon Cascade Lake.

Οι Optane DIMMs χρησιμοποιούν το πρωτόκολλο DDR-T, παρόμοιο με το DDR4, αλλά με μεταβλητή καθυστέρηση. Διαθέτουν ενσωματωμένο ελεγκτή για μετάφραση διευθύνσεων και εξισορρόπηση φθοράς.

Δεν απαιτούν ανανέωση δεδομένων όπως η DRAM, αλλά εμφανίζουν αύξηση των εγγραφών λόγω του μεγέθους πρόσβασης των 256 byte.

Σύγκριση Απόδοσης

Λειτουργία	**Καθυστέρηση (ns)**	**Εύρος Ζώνης (GB/s)**
DRAM (Ανάγνωση)	81	39.4
DRAM (Εγγραφή)	86	13.9
Optane (Ανάγνωση)	305	6.6
Optane (Εγγραφή)	94	2.3

Πίνακας 1: Καθυστέρηση και Εύρος Ζώνης DRAM και Optane [9]

Η Optane είναι πιο αργή από τη DRAM, αλλά προσφέρει διατήρηση δεδομένων, υψηλή πυκνότητα και χαμηλότερο κόστος.

Memkind API

Το Memkind είναι μια βιβλιοθήκη ανοικτού κώδικα από την Intel που προσφέρει ευέλικτο και αποδοτικό πλαίσιο κατανομής μνήμης για εφαρμογές που χρησιμοποιούν διάφορους τύπους μνήμης, όπως πτητική και persistent μνήμη. Υποστηρίζει κατανομή μνήμης από πηγές όπως DRAM και Intel Optane DC Persistent Memory, με υψηλή απόδοση και χαμηλό υπολογιστικό φόρτο.

Το Memkind επεκτείνει τις συναρτήσεις της γλώσσας ISO C, επιτρέποντας στους χρήστες να καθορίζουν τον τύπο μνήμης για κάθε λειτουργία:

- `void *memkind_malloc(memkind_t kind, size_t size)`: Δεσμεύει `size` bytes μνήμης τύπου `kind`.
- `void *memkind_calloc(memkind_t kind, size_t num, size_t size)`: Δεσμεύει μνήμη για `num` αντικείμενα των `size` bytes, αρχικοποιημένη σε μηδενικά.
- `void *memkind_realloc(memkind_t kind, void *ptr, size_t size)`: Αλλάζει το μέγεθος της μνήμης που δείχνει ο δείκτης `ptr`.
- `void memkind_free(memkind_t kind, void *ptr)`: Αποδεσμεύει τη μνήμη που δείχνει ο δείκτης `ptr`.

Η βιβλιοθήκη προσφέρει επίσης συναρτήσεις για τη δημιουργία και καταστροφή τύπων μνήμης:

- `int memkind_create_kind(memkind_memtype_t memtype_flags, memkind_policy_t policy, memkind_bits_t flags, memkind_t *kind)`: Δημιουργεί ένα `kind` που δεσμεύει μνήμη με συγκεκριμένο τύπο μνήμης, πολιτική δέσμευσης και σημαίες.
- `int memkind_destroy_kind(memkind_t kind)`: Καταστρέφει ένα `kind` που είχε δημιουργηθεί με `memkind_create_kind()`.

Μοντέλα Μηχανικής Μάθησης

Ακολουθούν τα μοντέλα μηχανικής μάθησης που χρησιμοποιήθηκαν στην παρούσα εργασία:

- **Δέντρο Απόφασης (Decision Tree)**: Το μοντέλο δέντρου απόφασης δημιουργεί ένα δέντρο όπου κάθε κόμβος περιλαμβάνει μια συνθήκη ενός χαρακτηριστικού και τα φύλλα του δέντρου περιλαμβάνουν τις τελικές κατηγορίες των δεδομένων.
- **Τυχαίο Δάσος (Random Forest)**: Το τυχαίο δάσος αποτελείται από πολλά δέντρα απόφασης, όπου οι προβλέψεις τους συνδυάζονται μέσω ψηφοφορίας για την αύξηση της ακρίβειας και τη μείωση της υπερεκπαίδευσης.

- **Κ-Πλησιέστεροι Γείτονες (KNN)**: Ο αλγόριθμος KNN ταξινομεί τα δεδομένα μέσω της ψηφοφορίας των κοντινότερων γειτόνων ενός σημείου, χρησιμοποιώντας την απόσταση για τον υπολογισμό της ομοιότητας μεταξύ των δεδομένων.
- **Naive Bayes**: Ο αλγόριθμος Naive Bayes βασίζεται στο θεώρημα του Bayes για να υπολογίσει τις πιθανότητες κατηγοριών, κάνοντάς τον γρήγορο και αποτελεσματικό, ιδανικό για δεδομένα με πολλές διαστάσεις.
- **Μηχανή Διανυσμάτων Υποστήριξης (SVM)**: Η SVM βρίσκει το καλύτερο επίπεδο για τη διαχωρισμό των δεδομένων σε κατηγορίες, χρησιμοποιώντας πυρήνες για την επίλυση προβλημάτων με μη γραμμική διαχωρισιμότητα.
- **Ανάλυση Κύριων Συνιστωσών (PCA)**: Η PCA μειώνει την πολυπλοκότητα των δεδομένων με την επιλογή των κύριων συνιστωσών που δημιουργούν τη μεγαλύτερη διακύμανση, βελτιώνοντας τη δυνατότητα ανάλυσης και επιτάχυνσης των υπολογιστικών διαδικασιών.

Μεθοδολογία και Πειραματική Αξιολόγηση

Αυτό το κεφάλαιο περιγράφει τη μεθοδολογία που χρησιμοποιείται σε αυτήν την έρευνα για την κατηγοριοποίηση των benchmarks με χρήση τεχνικών μηχανικής μάθησης και την αξιολόγηση της κατηγοριοποίησής τους με έναν αλγόριθμο πολλαπλών εργασιών. Η μεθοδολογία αποτελείται από διάφορα βασικά στάδια, όπως η συλλογή δεδομένων, η εξαγωγή χαρακτηριστικών, η κατηγοριοποίηση, η προγνωστική μοντελοποίηση και η αξιολόγηση. Κάθε στάδιο παίζει σημαντικό ρόλο στην εξασφάλιση της ακρίβειας και της ουσιαστικότητας της κατηγοριοποίησης.

Το πρώτο βήμα στη μεθοδολογία είναι **η ανάλυση προφίλ και ο χαρακτηρισμος** των benchmarks, όπου κάθε benchmark αναλύεται για την εξαγωγή σχετικών χαρακτηριστικών. Αυτά τα χαρακτηριστικά περιγράφουν τις ιδιότητες του benchmark, όπως μετρικές απόδοσης, υπολογιστική συμπεριφορά και χρήση πόρων. Μόλις ολοκληρωθεί η ανάλυση προφίλ, το επόμενο βήμα είναι η **κατηγοριοποίηση**, η οποία περιλαμβάνει την ομαδοποίηση των benchmarks σε κατηγορίες με βάση τα εξαγόμενα χαρακτηριστικά.

Μετά την κατηγοριοποίηση, η έρευνα εφαρμόζει την **προγνωστική μοντελοποίηση**, με στόχο την ανάπτυξη μοντέλων μηχανικής μάθησης που μπορούν να καθορίσουν αυτόματα την κατηγορία ενός νέου benchmark. Δοκιμάζονται διάφορα μοντέλα μηχανικής μάθησης, όπως τυχαία δάση, ο πλησιέστερος γείτονας (K-Nearest Neighbors) και τα μοντέλα Naive Bayes. Ο στόχος είναι να εντοπιστεί το πιο αποτελεσματικό μοντέλο για την κατηγοριοποίηση, βασιζόμενο στην ακρίβειά του.

Για την περαιτέρω αξιολόγηση της ακρίβειας της ταξινόμησης, προτείνεται ένας **αλγόριθμος τοποθέτησης βασισμένος στις κατηγορίες**. Ο αλγόριθμος αυτός αναθέτει τις εισερχόμενες διεργασίες σε τύπο μνήμης (DRAM ή PMEM) με βάση την κατηγορία στην οποία ανήκουν - όπως αυτή έχει προβλεφθεί από το μοντέλο μηχανικής μάθησης - και την τρέχουσα κατάσταση του συστήματος. Στόχος είναι η έξυπνη κατανομή των διεργασιών ώστε να αξιοποιούνται τα πλεονεκτήματα κάθε τύπου μνήμης. Η απόδοση του προτεινόμενου αλγορίθμου συγκρίνεται με αυτήν βασικών αλγορίθμων χρονοπρογραμματισμού, προκειμένου να αξιολογηθεί η αποτελεσματικότητά του και η επίδρασή του στη συνολική απόδοση του συστήματος.

Ανάλυση Προφίλ και Χαρακτηρισμός

Η ανάλυση προφίλ και ο χαρακτηρισμός των benchmarks αποτελεί το θεμελιώδες στάδιο της μεθοδολογίας, όπου κάθε benchmark αναλύεται διεξοδικά για την εξαγωγή σχετικών χαρακτηριστικών που θα χρησιμοποιηθούν για την κατηγοριοποίηση. Ο κύριος στόχος αυτού του σταδίου είναι η δημιουργία ενός συνόλου δεδομένων που να αποτυπώνει μια ευρεία ποικιλία συμπεριφορών benchmarks, διασφαλίζοντας ότι τα παραγόμενα δεδομένα αντικατοπτρίζουν διάφορα χαρακτηριστικά απόδοσης και υπολογιστικά μοτίβα.

Για την κατασκευή ενός συνόλου δεδομένων που καλύπτει ένα εύρος συμπεριφορών, αναπτύχθηκε μια σουίτα benchmarks, συνδυάζοντας προγράμματα από υφιστάμενες σουίτες benchmarks, όπως οι GAP,

Parsec, Rodinia και EEMBC. Αυτές οι σουίτες χρησιμοποιούνται ευρέως για την αξιολόγηση της απόδοσης των συστημάτων σε διάφορα σενάρια.

Με την επιλογή προγραμμάτων από πολλαπλές πηγές, το σύνολο δεδομένων αποτυπώνει ένα ευρύ φάσμα υπολογιστικών εργασιών, από επιστημονικούς υπολογισμούς και παράλληλες εφαρμογές έως benchmarks για ενσωματωμένα συστήματα.

Εκτός από τη συνδυασμένη χρήση benchmarks από διαφορετικές σουίτες, εισήχθησαν ποικίλες διαμορφώσεις εισόδου για την αύξηση του αριθμού των εργασιών και την ενίσχυση της ποικιλότητας των συμπεριφορών. Αυτές οι παραλλαγές σχεδιάστηκαν για να προάγουν τη δυσκολία των μοντέλων ταξινομητών, ενσωματώνοντας ένα ευρύ φάσμα προτύπων χρήσης πόρων, χρόνων εκτέλεσης και υπολογιστικών απαιτήσεων. Αυτή η προσέγγιση διασφαλίζει ότι το τελικό σύνολο δεδομένων είναι πλούσιο σε συμπεριφορές, καθιστώντας το κατάλληλο για την αξιολόγηση της αποτελεσματικότητας των μοντέλων μηχανικής μάθησης στην κατηγοριοποίηση διαφορετικών τύπων benchmarks.

Η σουίτα benchmarks εκτελέστηκε δύο φορές: μία χρησιμοποιώντας τα DRAM memory chips και μία χρησιμοποιώντας μόνο τις μονάδες Optane DIMMs της Intel. Αυτή η ρύθμιση επέτρεψε μια συγκριτική ανάλυση της απόδοσης των benchmarks υπό διαφορετικές αρχιτεκτονικές μνήμης. Ο χρόνος εκτέλεσης καταγράφηκε για κάθε δοκιμή, ενώ το Intel Performance Counter Monitor (PCM) χρησιμοποιήθηκε για τη συλλογή λεπτομερών μετρήσεων απόδοσης, παρέχοντας πληροφορίες σχετικά με τη χρήση των πόρων. Για κάθε δοκιμή, οι ακόλουθες μετρήσεις καταγράφηκαν ξεχωριστά:

- **Χρόνος εκτέλεσης**: Ο συνολικός χρόνος ολοκλήρωσης του benchmark, υποδεικνύοντας τη συνολική απόδοση για κάθε διαμόρφωση μνήμης.
- **Εντολές ανά κύκλο (IPC) με την πάροδο του χρόνου**: Μέτρο της χρήσης της CPU, δείχνοντας τον αριθμό των εκτελεσμένων εντολών ανά κύκλο ρολογιού καθ' όλη τη διάρκεια της εκτέλεσης.
- **Αναλογία επιτυχιών στην cache L3 με την πάροδο του χρόνου**: Το ποσοστό των προσβάσεων στη μνήμη cache που οδηγούν σε επιτυχία στην L3 cache, παρέχοντας πληροφορίες για τη μείωση της καθυστέρησης μνήμης.
- **Αστοχίες στην cache L3 με την πάροδο του χρόνου**: Ο αριθμός των περιπτώσεων όπου τα δεδομένα δεν βρέθηκαν στην cache L3.
- **Ρυθμός ανάγνωσης μνήμης (Read BW) με την πάροδο του χρόνου**: Ο ρυθμός με τον οποίο διαβάζονται δεδομένα από τη μνήμη, υποδεικνύοντας την αποδοτικότητα των αναγνώσεων κατά την εκτέλεση.
- **Ρυθμός εγγραφής μνήμης (Write BW) με την πάροδο του χρόνου**: Ο ρυθμός εγγραφής δεδομένων στη μνήμη, δείχνοντας την αποδοτικότητα των εγγραφών κατά την εκτέλεση.
- **Συνολικές προσβάσεις ανάγνωσης μνήμης**: Ο συνολικός αριθμός προσβάσεων ανάγνωσης μνήμης κατά την εκτέλεση στο Optane. Αυτό το μέτρο δεν εφαρμόζεται στο DRAM.
- **Συνολικές προσβάσεις εγγραφής μνήμης**: Ο συνολικός αριθμός προσβάσεων εγγραφής μνήμης στο Optane. Αυτό το μέτρο δεν εφαρμόζεται στο DRAM.

Αυτές οι μετρήσεις παρέχουν πολύτιμες πληροφορίες σχετικά με την απόδοση των benchmarks υπό διαφορετικές αρχιτεκτονικές μνήμης, επιτρέποντας μια ολοκληρωμένη κατανόηση της χρήσης των πόρων και της υπολογιστικής συμπεριφοράς. Τα εξαγόμενα χαρακτηριστικά είναι απαραίτητα για τη δημιουργία ακριβών μοντέλων μηχανικής μάθησης, ικανών να ταξινομούν benchmarks με βάση τα χαρακτηριστικά απόδοσής τους.

Κατηγοριοποίηση

Για την κατηγοριοποίηση των benchmarks και την προετοιμασία του συνόλου δεδομένων για τα μοντέλα μηχανικής μάθησης, καθορίζονται συγκεκριμένα όρια βάσει των μετρικών απόδοσης, ιδιαίτερα της χωρητικότητας μνήμης (Read BW και Write BW) και της χρησιμοποίησης CPU. Αυτά τα όρια βοηθούν

στην κατηγοριοποίηση των benchmarks σε μία από τις τέσσερις κατηγορίες: Memory Intensive, Read Intensive, Write Intensive και CPU Intensive.

Σημείωση: Οι μετρικές και τα όρια που αναφέρονται στη διαδικασία κατηγοριοποίησης αφορούν τις εκτελέσεις των benchmarks στην πλατφόρμα Optane, καθώς τα χαρακτηριστικά απόδοσης στην Optane ήταν καθοριστικά για τον προσδιορισμό αυτών των ορίων.

Η κατηγοριοποίηση των benchmarks στις τέσσερις κατηγορίες συνοψίζονται ως εξής:

- **Memory Intensive:** Benchmarks με Read BW και Write BW και τα δύο πάνω από το επιτρεπτό όριο. Αυτό το όριο ορίζεται ως το μισό της θεωρητικής μέγιστης τιμής. Ο καθορισμός του ορίου στο μισό της μέγιστης τιμής Read και Write BW λαμβάνει υπόψη ότι κάποια benchmarks ενδέχεται να χρησιμοποιούν τη χωρητικότητα μνήμης στο πλήρες της δυναμικό μόνο για μέρος του χρόνου εκτέλεσης. Σε αυτές τις περιπτώσεις, κατά τη διάρκεια του υπόλοιπου χρόνου εκτέλεσης, η χρησιμοποίηση της μνήμης μπορεί να μειωθεί σημαντικά ή ακόμη και να παραμείνει αδρανής. Παρόλα αυτά, αυτά τα προγράμματα εξακολουθούν να εμφανίζουν σημαντική ζήτηση μνήμης κατά τις περιόδους υψηλής χρησιμοποίησης, γεγονός που δικαιολογεί την κατηγοριοποίησή τους ως memory intensive.
- **Read Intensive:** Περιλαμβάνει τα benchmarks στα οποία το Read BW υπερβαίνει το καθορισμένο όριο, αλλά το Write BW είναι σχετικά χαμηλό. Αυτά τα benchmarks επικεντρώνονται στις αναγνώσεις μνήμης, πράγμα που σημαίνει ότι απαιτούν σημαντική ροή ανάγνωσης, αλλά δεν εγγράφουν μεγάλες ποσότητες δεδομένων στη μνήμη.
- **Write Intensive:** Περιλαμβάνει τα benchmarks όπου το Write BW υπερβαίνει το όριο, ενώ το Read BW παραμένει σχετικά χαμηλό. Αυτά τα benchmarks χαρακτηρίζονται από μια υψηλή συχνότητα λειτουργιών εγγραφής στη μνήμη, απαιτώντας σημαντική ροή εγγραφής.
- **CPU Intensive:** Benchmarks που δεν πληρούν κανένα από τα παραπάνω κριτήρια και χρησιμοποιούν κυρίως την CPU. Τα CPU Intensive benchmarks επικεντρώνονται περισσότερο σε υπολογιστικά καθήκοντα και εμφανίζουν υψηλότερο IPC από τα benchmarks άλλων κατηγοριών, ενώ η χρήση μνήμης τους τείνει να είναι χαμηλότερη.

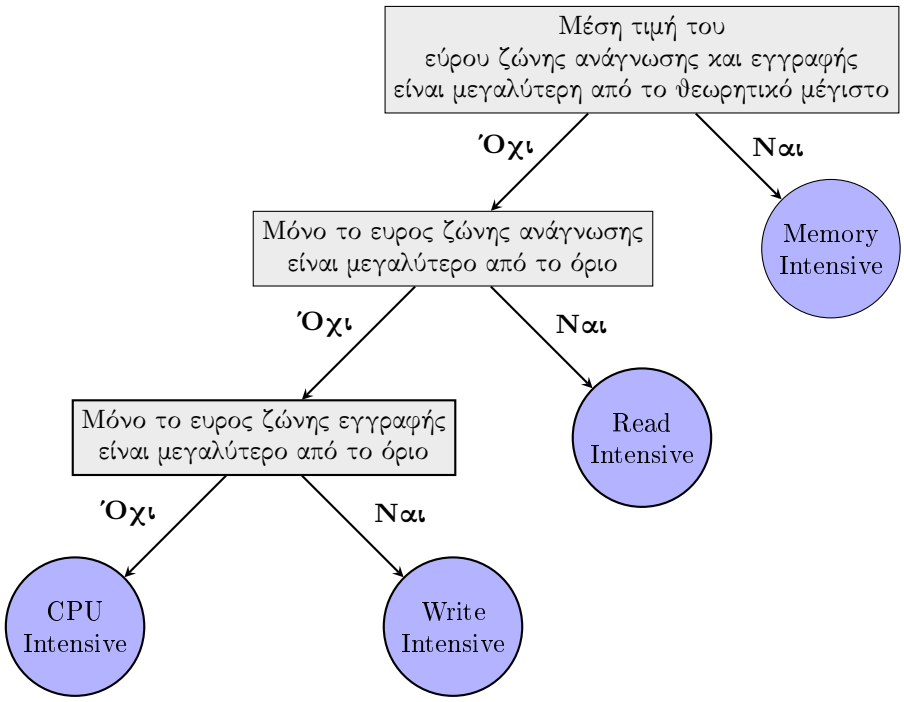
Αυτά τα κριτήρια επιτρέπουν την κατηγοριοποίηση κάθε benchmark με βάση τη χρήση μνήμης και CPU, δημιουργώντας ένα σύνολο δεδομένων με ετικέτες που μπορεί να χρησιμοποιηθεί για την εκπαίδευση μοντέλων μηχανικής μάθησης για την πρόβλεψη της συμπεριφοράς νέων benchmarks.

Προσέγγιση Εποπτευόμενης Μάθησης

Τώρα που έχει δημιουργηθεί ένα σύνολο δεδομένων με ετικέτες βασισμένο στην κατηγοριοποίηση των benchmarks σε διακριτές κατηγορίες (Memory Intensive, Read Intensive, Write Intensive, και CPU Intensive), το επόμενο βήμα είναι η εφαρμογή μοντέλων μηχανικής μάθησης για την πρόβλεψη της κατηγορίας νέων, αόρατων benchmarks. Αυτή η διαδικασία περιλαμβάνει την εκπαίδευση εποπτευόμενων μοντέλων μηχανικής μάθησης χρησιμοποιώντας το σύνολο δεδομένων με ετικέτες, ακολουθούμενη από την αξιολόγηση της απόδοσης τους.

Ένας μεγάλος αριθμός χαρακτηριστικών περιλαμβάνεται στο σύνολο δεδομένων για να αξιολογηθεί αν τα μοντέλα μπορούν να εντοπίσουν σωστά τις κατηγορίες των benchmarks ανεξαρτήτως των συγκεκριμένων χαρακτηριστικών που χρησιμοποιούνται. Σε ένα άλλο σενάριο, η κατηγοριοποίηση μπορεί να εξαρτάται από διαφορετικά χαρακτηριστικά πέρα από τη χωρητικότητα μνήμης, απαιτώντας μια πιο ευέλικτη προσέγγιση. Αντί να βασιστεί αποκλειστικά σε ένα προκαθορισμένο σύνολο βασικών μετρικών, η μεθοδολογία που προτείνεται σε αυτήν την έρευνα είναι σχεδιασμένη να λειτουργεί σε διαφορετικές υλοποιήσεις, επιτρέποντας προσαρμοστικότητα σε διάφορες εργασίες κατηγοριοποίησης.

Δεδομένου ότι πολλές από τις μετρικές χαρτογράφησης συλλέγονται ως δεδομένα χρονοσειρών, η μέση τιμή κάθε μετρικής χρησιμοποιείται ως αντιπροσωπευτικό χαρακτηριστικό. Ενώ τα πλήρη χρονικά διαγράμματα θα μπορούσαν να παρέχουν πιο λεπτομερείς πληροφορίες, η ενσωμάτωσή τους άμεσα στα μοντέλα μηχανικής μάθησης θα απαιτούσε περίπλοκες αρχιτεκτονικές ικανές να επεξεργάζονται ακολουθιακά δεδομένα. Η χρήση μέσων τιμών απλοποιεί την υλοποίηση, ενώ εξακολουθεί να καταγράφει τις γενικές τάσεις στη συμπεριφορά των benchmarks.



Εικόνα 2: Διάγραμμα των κριτηρίων κατηγοριοποίησης των benchmarks.

Τα χαρακτηριστικά που εξάγονται για την κατηγοριοποίηση είναι:

- **Χαρακτηριστικά που εξάγονται κατά την εκτέλεση σε DRAM:**
  - Χρόνος εκτέλεσης
  - Μέση τιμή των εντολών ανά Κύκλο (IPC)
  - Μέση τιμή του ποσοστού επιτυχίας L3 cache
  - Μέση τιμή των αποτυχιών L3 cache ανά δευτερόλεπτο
  - Μέση τιμή του ρυθμού ανάγνωσης μνήμης (Read BW)
  - Μέση τιμή του ρυθμού εγγραφής μνήμης (Write BW)
- **Χαρακτηριστικά που εξάγονται κατά την εκτέλεση σε Optane:**
  - Χρόνος εκτέλεσης
  - Μέση τιμή των εντολών ανά Κύκλο (IPC)
  - Μέση τιμή του ποσοστού επιτυχίας L3 cache
  - Μέση τιμή των αποτυχιών L3 cache ανά δευτερόλεπτο
  - Μέση τιμή του ρυθμού ανάγνωσης μνήμης (Read BW)
  - Μέση τιμή του ρυθμού εγγραφής μνήμης (Write BW)
  - Συνολικές αναγνώσεις μνήμης
  - Συνολικές εγγραφές μνήμης

Λόγω του υψηλού αριθμού διαστάσεων του συνόλου χαρακτηριστικών, εφαρμόζονται τεχνικές μείωσης διάστασης για να βελτιωθεί η απόδοση των μοντέλων.

Μείωση Διάστασης

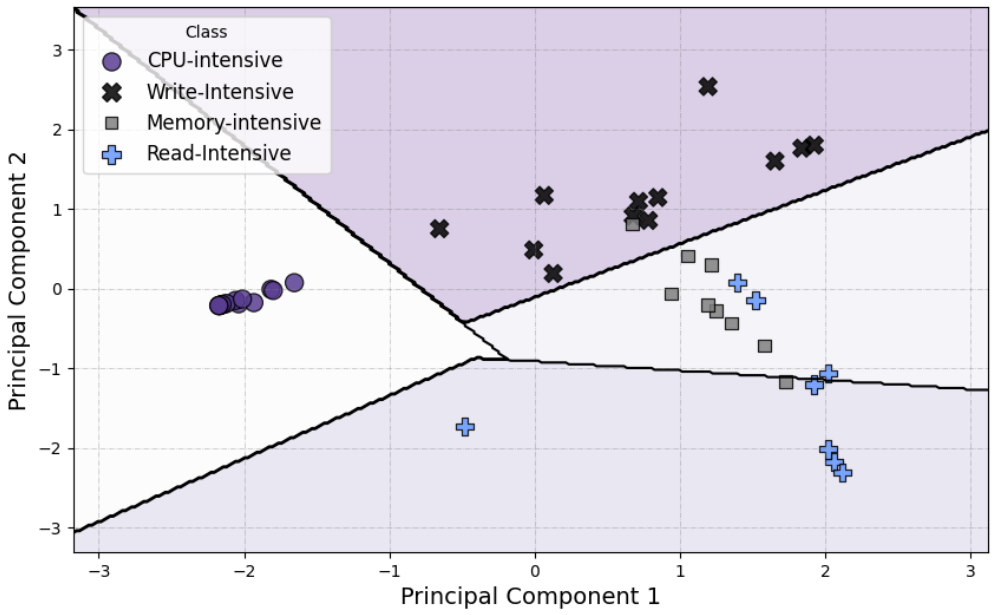
Δεδομένου ότι τα μοντέλα μηχανικής μάθησης αποδίδουν καλύτερα με δεδομένα χαμηλότερης διάστασης, εφαρμόζεται μείωση διάστασης για να μειωθεί ο χώρος χαρακτηριστικών πριν την εκπαίδευση των μοντέλων. Αυτό είναι ιδιαίτερα σημαντικό επειδή πολλά από τα αρχικά χαρακτηριστικά είναι αλληλοσυσχετιζόμενα, κάτι που μπορεί να μειώσει την αποτελεσματικότητα ορισμένων μοντέλων.

Για να επιτευχθεί αυτό, χρησιμοποιείται το μοντέλο Random Forest για να προσδιοριστεί η σημασία των χαρακτηριστικών στη διαδικασία αυτή. Συγκεκριμένα, επιλέγονται τα πιο σημαντικά χαρακτηριστικά έτσι ώστε η συνδυασμένη τους σημασία να καλύπτει το 70% της συνολικής σημαντικότητας των χαρακτηριστικών. Αυτό βοηθά στην διατήρηση των πιο σημαντικών πληροφοριών ενώ απορρίπτονται λιγότερο σημαντικά χαρακτηριστικά.

Συγκεκριμένα, τα πιο σημαντικά χαρακτηριστικά που εντοπίστηκαν ήταν το εύρος ζώνης ανάγνωσης στην DRAM, οι αστοχίες της κρυφής μνήμης L3 και το εύρος ζώνης ανάγνωσης και εγγραφής στην Optane. Αυτό είναι σημαντικό, καθώς δύο από αυτά τα τέσσερα χαρακτηριστικά (το εύρος ζώνης ανάγνωσης και εγγραφής στην Optane) χρησιμοποιήθηκαν στα κριτήρια ταξινόμησης για την κατηγοριοποίηση των benchmarks σε Memory-intensive, CPU-intensive και άλλες κατηγορίες. Αυτό καταδεικνύει την αποτελεσματικότητα της μεθόδου, καθώς κατάφερε με επιτυχία να απορρίψει μη σχετικές πληροφορίες και να διατηρήσει τα πιο σημαντικά χαρακτηριστικά.

Για περαιτέρω μείωση, εφαρμόζεται η Ανάλυση Κύριων Συνιστωσών (PCA) στα επιλεγμένα σημαντικά χαρακτηριστικά. Η PCA προβάλει τα δεδομένα σε έναν χαμηλότερης διάστασης χώρο, διατηρώντας τη μέγιστη διακύμανση των δεδομένων. Σε αυτή την περίπτωση, η διάσταση μειώνεται σε δύο, απλοποιώντας το πρόβλημα και επιτρέποντας στα μοντέλα να αποδώσουν καλύτερα με λιγότερα χαρακτηριστικά.

Μετά την εφαρμογή της Ανάλυσης Κύριων Συνιστωσών (PCA), το σύνολο δεδομένων μειώθηκε σε δύο διαστάσεις. Αυτό επέτρεψε μια σαφή οπτική αναπαράσταση των κατηγοριών των benchmarks στον μειωμένο χώρο (Εικόνα 3). Το παρακάτω σχήμα δείχνει την απεικόνιση PCA, όπου κάθε σημείο αντιπροσωπεύει ένα benchmark. Το χρώμα των σημείων υποδεικνύει την κατηγορία τους και είναι εμφανές ότι τα benchmarks κατανέμονται σε διακριτές περιοχές μέσα στον διδιάστατο χώρο.



Εικόνα 3: Οπτικοποίηση PCA του Συνόλου Δεδομένων και Ταξινόμηση SVM

Από το γράφημα, μπορούμε παρατηρείται ότι οι διαφορετικές κατηγορίες ομαδοποιούνται σε ξεχωριστές περιοχές. Αυτή η κατανομή υποδηλώνει ότι τα benchmarks της ίδιας κατηγορίας μοιράζονται παρόμοια χαρακτηριστικά και αυτά τα μοτίβα είναι εμφανή στο γράφημα PCA. Το μοντέλο SVM, όταν εφαρμόστηκε στον μειωμένο διδιάστατο χώρο, μπόρεσε εύκολα να χαράξει γραμμικά όρια μεταξύ των κατηγοριών. Οι καλά καθορισμένες περιοχές καθιστούν ευκολότερο για τον ταξινομητή να διακρίνει τις κατηγορίες με βάση τα εξαγόμενα χαρακτηριστικά.

Αυτές οι παρατηρήσεις δείχνουν ότι το PCA κατάφερε να αποτυπώσει επιτυχώς την υποκείμενη δομή των δεδομένων σε μόλις δύο διαστάσεις και ότι το μοντέλο SVM αποδίδει καλά στον διαχωρισμό των benchmarks στις αντίστοιχες κατηγορίες τους. Αυτό παρέχει την ένδειξη ότι και άλλα μοντέλα, όπως το Random Forest, το KNN και το Naive Bayes, πιθανότατα θα αποδώσουν καλά με αυτή τη μείωση διαστάσεων, καθώς τα δεδομένα βρίσκονται πλέον σε έναν χώρο που διακρίνει σαφώς τις διαφορετικές κατηγορίες.

Μοντέλα Μηχανικής Μάθησης για Πρόβλεψη Κατηγοριοποίησης

Για αυτή τη μελέτη, επιλέχθηκαν τα εξής εποπτευόμενα μοντέλα μάθησης:

- **Random Forests**: Ο αριθμός των δέντρων ήταν 1000, με το κριτήριο διαίρεσης να είναι το 2, και το ελάχιστο μέγεθος κόμβου να είναι 1.
- **K-Nearest Neighbors (KNN)**: Η τιμή του k ορίστηκε σε 4, χρησιμοποιώντας το μέτρο απόστασης Minkowski, και δεν εφαρμόστηκε βάρος.
- **Naive Bayes**

Αυτά τα μοντέλα επιλέχθηκαν για την ικανότητά τους να χειρίζονται αποτελεσματικά εργασίες κατηγοριοποίησης και είναι καλά προσαρμοσμένα στην εποπτευόμενη κατηγοριοποίηση. Για να διασφαλιστούν αξιόπιστα αποτελέσματα, η πλήρης διαδικασία εκπαίδευσης και δοκιμών επαναλαμβάνεται 10 φορές για κάθε διαχωρισμό του συνόλου δεδομένων και εκπαίδευσης. Αυτό μετριάζει την επίδραση της τυχαιότητας στην επιλογή των δεδομένων. Οι τελικές μετρικές αξιολόγησης προκύπτουν από τον μέσο όρο των αποτελεσμάτων σε όλες τις επαναλήψεις. Κάθε επανάληψη χρησιμοποιεί έναν διαχωρισμό του συνόλου δεδομένων με 35 δείγματα εκπαίδευσης και 10 δείγματα δοκιμών.

Το πρώτο βήμα περιλάμβανε την εκπαίδευση των μοντέλων χρησιμοποιώντας το αρχικό σύνολο δεδομένων. Η απόδοση αξιολογήθηκε χρησιμοποιώντας μόνο τη μετρική της ακρίβειας. Αφού χρησιμοποιήθηκε το Random Forest για τη μείωση του συνόλου δεδομένων, τα μοντέλα επανεκπαιδεύτηκαν στο μειωμένο σύνολο χαρακτηριστικών. Στη συνέχεια, εφαρμόστηκε PCA για τη μείωση της διάστασης του συνόλου δεδομένων σε δύο, και τα μοντέλα επανεκπαιδεύτηκαν στο μειωμένο μέσω PCA σύνολο δεδομένων.

Τα μοντέλα αξιολογήθηκαν, δηλαδή, σε τρία διαφορετικά σενάρια: το αρχικό σύνολο δεδομένων, το σύνολο δεδομένων με τα σημαντικά χαρακτηριστικά και το σύνολο δεδομένων μειωμένο μέσω PCA. Ο μέσος όρος ακρίβειας για κάθε ένα από αυτά τα σενάρια υπολογίστηκε και τα αποτελέσματα συνοψίζονται στον παρακάτω πίνακα.

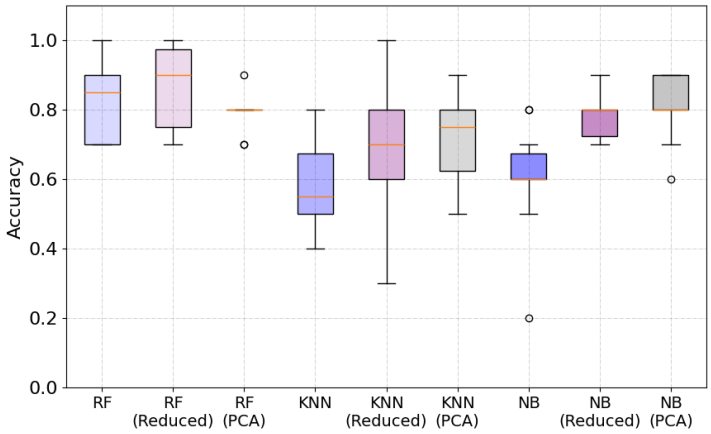
Σενάριο	**Random Forest**	**KNN**	**Naive Bayes**
Αρχικό Σύνολο Δεδομένων	82%	58%	60%
Σύνολο Σημαντικών Χαρακτηριστικών	87%	68%	79%
Σύνολο Μειωμένο μέσω PCA	79%	72%	81%

Πίνακας 2: Μέσες τιμές ακρίβειας για όλα τα σενάρια αξιολόγησης.

Τα αποτελέσματα του Πίνακα 2 αποκαλύπτουν αρκετές σημαντικές τάσεις. Πρώτον, το μοντέλο Random Forest είχε σταθερά καλή απόδοση, επιτυγχάνοντας ακρίβεια 82% στο αρχικό σύνολο δεδομένων, η οποία αυξήθηκε σε 87% μετά την επιλογή σημαντικών χαρακτηριστικών και παρέμεινε υψηλή στο 79% μετά τη μείωση μέσω PCA. Αντίθετα, το μοντέλο KNN αρχικά παρουσίασε δυσκολίες στο αρχικό σύνολο

δεδομένων με ακρίβεια 58%, αλλά η απόδοσή του βελτιώθηκε σημαντικά σε 68% όταν χρησιμοποιήθηκαν μόνο τα σημαντικά χαρακτηριστικά και περαιτέρω σε 72% μετά τη μείωση μέσω PCA. Ομοίως, το Naive Bayes σημείωσε βελτίωση από 60% στο αρχικό σύνολο δεδομένων σε 79% με την επιλογή σημαντικών χαρακτηριστικών και στη συνέχεια σε 81% με τη μείωση μέσω PCA.

Αυτές οι παρατηρήσεις δείχνουν ότι η μείωση του χώρου χαρακτηριστικών - αρχικά μέσω της επιλογής των πιο σχετικών χαρακτηριστικών και στη συνέχεια μέσω της εφαρμογής του PCA - όχι μόνο βοηθά στην εξάλειψη μη σχετικών πληροφοριών αλλά και βελτιώνει σημαντικά την απόδοση των μοντέλων που είναι πιο ευαίσθητα σε δεδομένα υψηλής διαστατικότητας. Ειδικότερα, η αξιοσημείωτη βελτίωση για τα KNN και Naive Bayes υποδηλώνει ότι αυτή η μέθοδος μείωσης καταφέρνει να αποτυπώσει αποτελεσματικά τη δομή των δεδομένων.



Εικόνα 4: Διαγράμματα box plot της ακρίβειας των μοντέλων και των συνόλων δεδομένων.

Με βάση τις παρουσιασμένες τιμές ακρίβειας, η καλύτερη απόδοση επιτεύχθηκε από το μοντέλο Random Forest στο σύνολο δεδομένων με τα σημαντικά χαρακτηριστικά, με ακρίβεια 87%. Συνεπώς, το μοντέλο Random Forest εκπαιδευμένο στο σύνολο δεδομένων με τα σημαντικά χαρακτηριστικά θα χρησιμοποιηθεί για περαιτέρω αξιολόγηση με τον πολυεργασιακό αλγόριθμο.

Αξιολόγηση με Χρήση Αλγορίθμου Βασισμένου σε Κατηγορίες

Σε αυτή την ενότητα, επιδιώκεται η αξιολόγηση της αποτελεσματικότητας της μεθοδολογίας κατηγοριοποίησης βάσει κλάσεων που αναπτύχθηκε στα προηγούμενα κεφάλαια. Για τον σκοπό αυτό, προσομοιώνεται ένα περιβάλλον πολυδιεργασίας, όπου πολλαπλά benchmarks εκτελούνται ταυτόχρονα στο σύστημα.

Δημιουργείται μια ουρά από benchmarks, με κάθε benchmark να εισάγεται στο σύστημα σύμφωνα με μια κατανομή καθυστέρησης. Δοκιμάζονται διαφορετικές κατανομές καθυστέρησης, αριθμοί ταυτόχρονων διεργασιών και αλγόριθμοι τοποθέτησης, ώστε να αξιολογηθεί πλήρως η συμπεριφορά του συστήματος.

Μεταξύ των στρατηγικών τοποθέτησης που αξιολογούνται, μία βασίζεται στην κατηγοριοποίηση κατά κλάσεις όπως αυτή προέκυψε από το μοντέλο μηχανικής μάθησης. Αυτή η προσέγγιση τοποθετεί τις διεργασίες έξυπνα, με βάση την προβλεπόμενη συμπεριφορά τους ως προς τους πόρους. Επιπλέον, υλοποιούνται διάφοροι βασικοί αλγόριθμοι τοποθέτησης για λόγους σύγκρισης, όπως τυχαία τοποθέτηση και τοποθέτηση τύπου round-robin.

Προτεινόμενος Αλγόριθμος Τοποθέτησης βάσει Κλάσεων

Ο Αλγόριθμος Τοποθέτησης βάσει Κλάσεων σχεδιάστηκε για την αποδοτική διαχείριση των διεργασιών, λαμβάνοντας υπόψη τα χαρακτηριστικά μνήμης κάθε προγράμματος. Ο αλγόριθμος αυτός αξιοποιεί την κατηγοριοποίηση των προγραμμάτων όπως αυτή προέκυψε από το μοντέλο που πέτυχε τη μεγαλύτερη ακρίβεια. Κάθε διεργασία ταξινομείται σε μία από τις 4 κατηγορίες.

Αφού ταξινομηθούν, οι διεργασίες ανατίθενται είτε στη μνήμη DRAM είτε στη μνήμη PMEM, με βάση τα χαρακτηριστικά τους και τις απαιτήσεις του κάθε τύπου μνήμης. Η λογική τοποθέτησης ακολουθεί ένα σύνολο κανόνων, με στόχο τη μεγιστοποίηση της απόδοσης του συστήματος μέσω της εξισορρόπησης της χρήσης μνήμης και της αξιοποίησης των πλεονεκτημάτων κάθε τύπου μνήμης.

Ο αλγόριθμος ακολουθεί ένα σύνολο ιεραρχημένων κανόνων και εκτελεί τον πρώτο κανόνα που ικανοποιείται:

- Αν η διεργασία έχει ταξινομηθεί ως **CPU-Intensive**, τότε ανατίθεται στη **PMEM**.
- Αν η διεργασία έχει ταξινομηθεί ως **Memory-Intensive**, τότε ανατίθεται στη **DRAM**.
- Αν ο αριθμός των ενεργών διεργασιών στην **PMEM** υπερβαίνει αυτόν της **DRAM**, η διεργασία ανατίθεται στη **DRAM** για την εξισορρόπηση του φορτίου.
- Αν ο αριθμός των ενεργών διεργασιών στη **DRAM** υπερβαίνει αυτόν της **PMEM**, η διεργασία ανατίθεται στη **PMEM**.
- Αν ο αριθμός των ενεργών διεργασιών σε **PMEM** και **DRAM** είναι ίσος, τότε η απόφαση βασίζεται στον τύπο της διεργασίας:
  - Οι **Read-Intensive** διεργασίες ανατίθενται στην **PMEM**, καθώς αυτή διαχειρίζεται καλύτερα τις αναγνώσεις και διατηρεί τη DRAM για πιο απαιτητικές διεργασίες.
  - Οι **Write-Intensive** διεργασίες ανατίθενται στη **DRAM**, η οποία προσφέρει καλύτερη απόδοση στις εγγραφές.

Αλγόριθμοι Τοποθέτησης

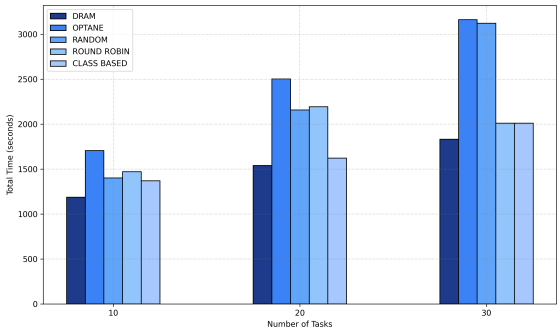
Στη μελέτη αυτή, αξιολογούνται πέντε διαφορετικοί αλγόριθμοι τοποθέτησης, καθένας από τους οποίους σχεδιάστηκε για να αναθέτει εργασίες σε έναν μόνο τύπο μνήμης - είτε DRAM είτε PMEM. Οι αλγόριθμοι διαφέρουν στον τρόπο με τον οποίο αναθέτουν τις εργασίες με βάση τις απαιτήσεις μνήμης των προγραμμάτων. Οι παρακάτω στρατηγικές τοποθέτησης αξιολογούνται:

- **Όλα στη DRAM**: Στο σενάριο αυτό, όλα τα προγράμματα ανατίθενται στη DRAM, ανεξαρτήτως των απαιτήσεων μνήμης τους.
- **Όλα στο PMEM**: Όλα τα προγράμματα ανατίθενται στο PMEM (Optane), ανεξάρτητα από τις απαιτήσεις μνήμης τους.
- **Τυχαία Ανάθεση**: Στην περίπτωση αυτή, οι εργασίες ανατίθενται τυχαία είτε στη DRAM είτε στο PMEM.
- **Round Robin**: Η μέθοδος τοποθέτησης Round Robin εναλλάσσεται ανάμεσα στην ανάθεση εργασιών στη DRAM και το PMEM με κυκλικό τρόπο.
- **Αλγόριθμος Βασισμένος σε Κατηγορίες**: Ο αλγόριθμος τοποθέτησης βασισμένος σε κατηγορίες.

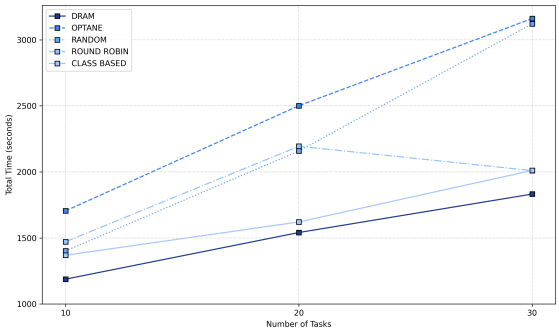
Τα πειράματα διεξήχθησαν χρησιμοποιώντας τρεις διαφορετικές κατανομές καθυστέρησης - **Ομοιόμορφη (Uniform)**, **Κανονική (Gaussian)** και **Poisson** - προκειμένου να προσομοιωθούν διαφορετικά πρότυπα άφιξης διεργασιών. Κάθε ένας από τους πέντε αλγορίθμους κατανομής εκτελέστηκε υπό αυτές τις κατανομές, με αποτέλεσμα συνολικά 15 πειραματικές δοκιμές. Σε κάθε πειραματικό σενάριο, μια παρτίδα από 10, 20 ή 30 benchmarks εισάγεται στο σύστημα σύμφωνα με την αντίστοιχη κατανομή καθυστέρησης.

Συνολικός Χρόνος Εκτέλεσης

Σε αυτό το πείραμα, συγκρίνουμε τους χρόνους εκτέλεσης μεταξύ όλων των αλγορίθμων κατανομής, λαμβάνοντας τον μέσο όρο των αποτελεσμάτων που προέκυψαν από τις διαφορετικές κατανομές καθυστέρησης.



Συνολικός Χρόνος Εκτέλεσης για Διαφορετικούς Αλγορίθμους Τοποθέτησης.

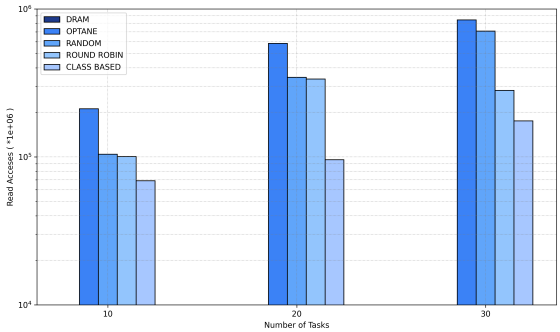


Συνολικός Χρόνος Εκτέλεσης σε Συνάρτηση με τον Αριθμό των Εργασιών.

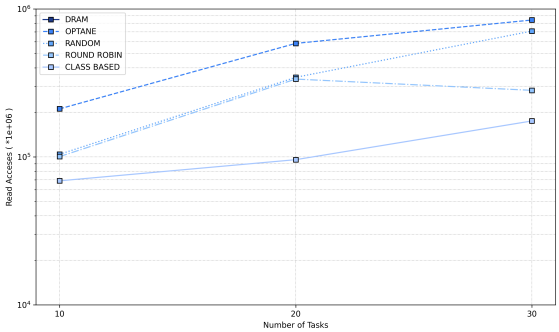
Εικόνα 5: Σύγκριση του συνολικού χρόνου εκτέλεσης μεταξύ των αλγορίθμων τοποθέτησης. Αριστερά: συγκεντρωτικοί συνολικοί χρόνοι. Δεξιά: Συμπεριφορά κλιμάκωσης με την αύξηση του αριθμού των εργασιών.

Προσβάσεις Ανάγνωσης και Εγγραφής

Ένας ακόμη σημαντικός δείκτης απόδοσης είναι ο αριθμός των προσβάσεων στη μνήμη - συγκεκριμένα, πόσο συχνά πραγματοποιούνται αναγνώσεις ή εγγραφές στην PMEM. Αυτές οι προσβάσεις επηρεάζουν σημαντικά τόσο τον χρόνο εκτέλεσης όσο και τη μακροχρόνια ανθεκτικότητα της μνήμης, ιδιαίτερα λόγω της χαμηλότερης ταχύτητας και της μειωμένης αντοχής σε εγγραφές της PMEM σε σύγκριση με τη DRAM.

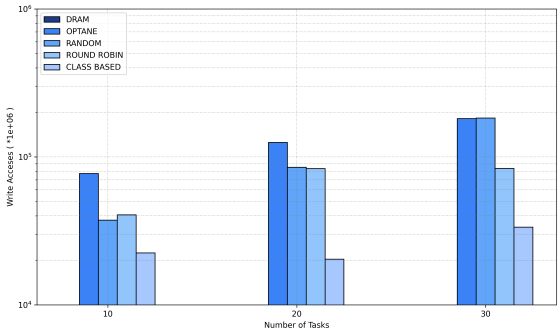


Συνολικές Προσβάσεις Ανάγνωσης ανάμεσα σε Αλγορίθμους Τοποθέτησης.

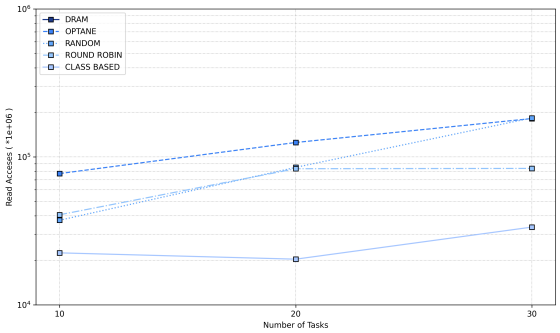


Προσβάσεις Ανάγνωσης vs Αριθμός Εργασιών

Εικόνα 6: Σύγκριση Προτύπων Πρόσβασης Ανάγνωσης. Αριστερά: Πώς οι Προσβάσεις Κλιμακώνονται με την Αύξηση του Αριθμού Εργασιών.



Συνολικές Προσβάσεις Εγγραφής ανάμεσα σε Αλγορίθμους Τοποθέτησης.

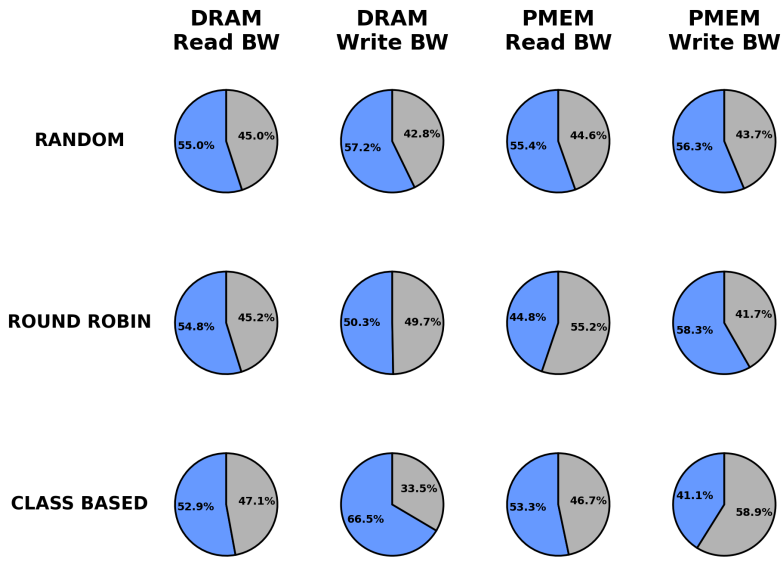


Προσβάσεις Εγγραφής vs Αριθμός Εργασιών.

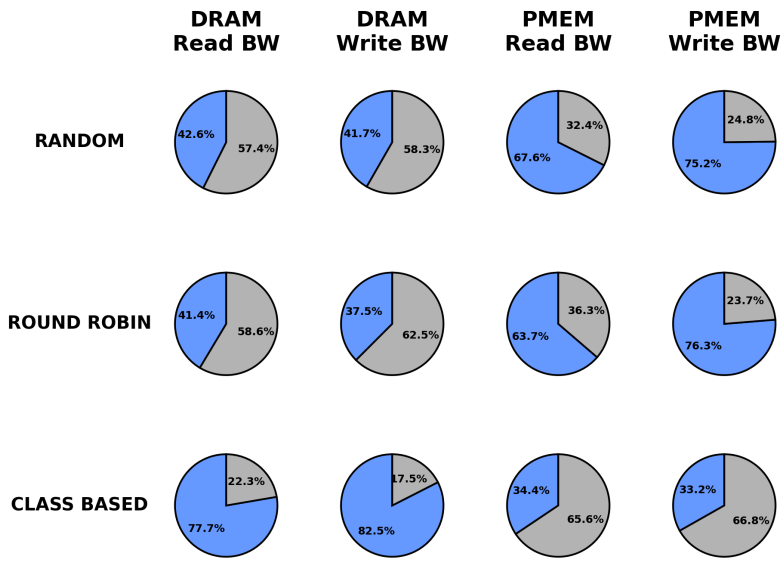
Εικόνα 7: Σύγκριση Προτύπων Πρόσβασης Εγγραφής. Αριστερά: Συγκεντρωμένες Προσβάσεις Εγγραφής. Δεξιά: Πώς οι Προσβάσεις Κλιμακώνονται με την Αύξηση του Αριθμού Εργασιών.

Χρήση Μνήμης

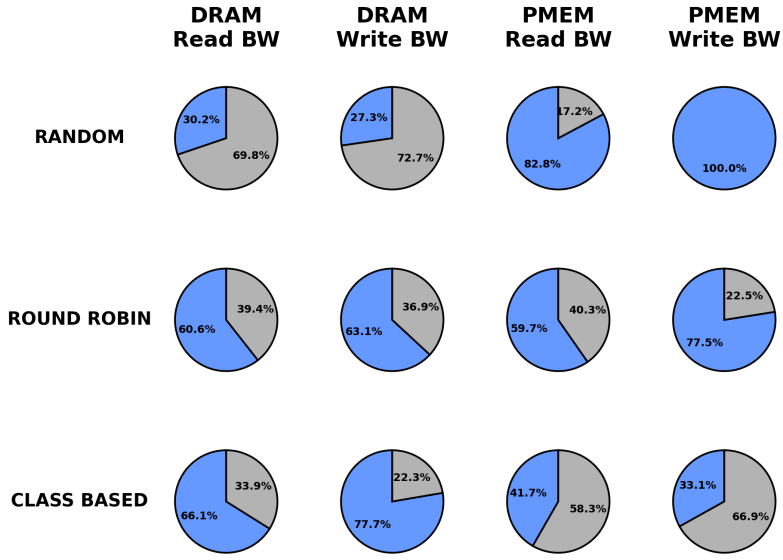
Η Χρήση Μνήμης παρέχει μια πιο λεπτομερή κατανόηση του πόσο αποτελεσματικά το σύστημα εκμεταλλεύεται τη διαθέσιμη χωρητικότητα μνήμης τόσο στη DRAM όσο και στην PMEM κατά τη διάρκεια της εκτέλεσης. Αυτός ο δείκτης χωρίζεται σε τέσσερα συστατικά: Εύρος Ζώνης Ανάγνωσης DRAM, Εύρος Ζώνης Εγγραφής DRAM, Εύρος Ζώνης Ανάγνωσης PMEM και Εύρος Ζώνης Εγγραφής PMEM.



Εικόνα 8: Χρήση Μνήμης (10 Εργασίες) ανάμεσα σε DRAM και PMEM για Αλγορίθμους Τοποθέτησης Τυχαία, Κυκλική και Βασισμένη σε Κλάσεις.



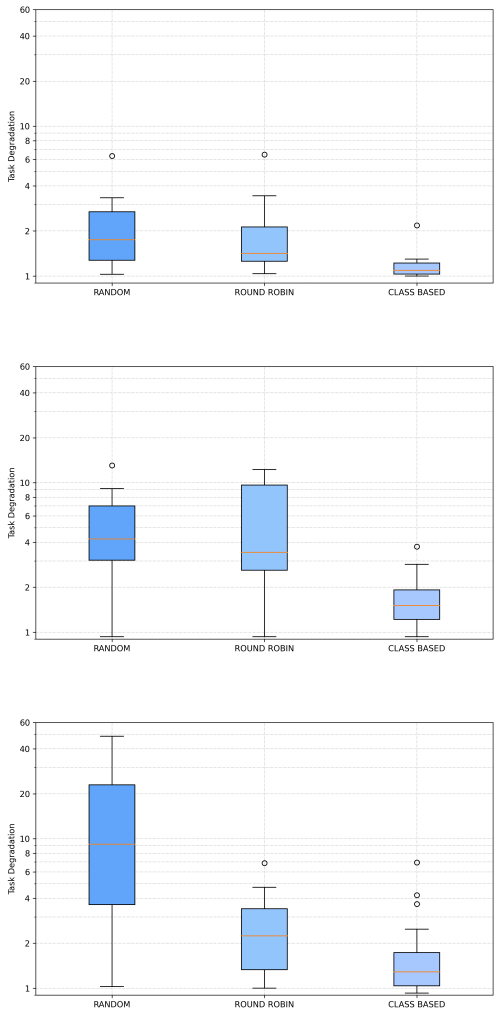
Εικόνα 9: Χρήση Μνήμης (20 Εργασίες) ανάμεσα σε DRAM και PMEM για Αλγορίθμους Τοποθέτησης Τυχαία, Κυκλική και Βασισμένη σε Κλάσεις.



Εικόνα 10: Χρήση Μνήμης (30 Εργασίες) ανάμεσα σε DRAM και PMEM για Αλγορίθμους Τοποθέτησης Τυχαία, Κυκλική και Βασισμένη σε Κλάσεις.

Παράπτωμα Χρόνου Εκτέλεσης Εργασίας

Το Παράπτωμα Χρόνου Εκτέλεσης Εργασίας παρέχει μια λεπτομερή άποψη του πώς κάθε αλγόριθμος τοποθέτησης επηρεάζει την απόδοση των μεμονωμένων εργασιών σε σχέση με τον ιδανικό τους χρόνο εκτέλεσης στη DRAM.



Εικόνα 11: Παράπτωμα Χρόνου Εκτέλεσης Εργασίας. Οι χρόνοι εκτέλεσης είναι κανονικοποιημένοι ως προς την εκτέλεση μόνο με DRAM. Η ανώτερη γραφική παράσταση αντιστοιχεί σε 10 εργασίες, η μεσαία σε 20 εργασίες, και η κατώτερη σε 30 εργασίες.

Περίληψη των Αποτελεσμάτων Αξιολόγησης

Η πειραματική αξιολόγηση δείχνει την αποτελεσματικότητα του προτεινόμενου **Αλγορίθμου Τοποθέτησης Βασισμένου σε Κατηγορίες**, ο οποίος χρησιμοποιεί ταξινόμηση για να λάβει τεκμηριωμένες αποφάσεις τοποθέτησης με βάση τα χαρακτηριστικά των εργασιών και την κατάσταση του συστήματος.

Ο Αλγόριθμος Τοποθέτησης Βασισμένος σε Κατηγορίες συγκρίθηκε με τέσσερις βασικούς αλγορίθμους αναφοράς: **Μόνο DRAM**, **Μόνο PMEM**, **Τυχαία Ανάθεση** και **Round Robin**.

Σε πολλαπλά μετρικά και πειραματικά σενάρια, η προσέγγιση Βασισμένη σε Κατηγορίες παρουσίασε σημαντικά πλεονεκτήματα:

- **Χρόνος Εκτέλεσης**: Ο αλγόριθμος Βασισμένος σε Κατηγορίες κατέγραψε σταθερά χαμηλότερους χρόνους εκτέλεσης σε σχέση με τους αλγορίθμους Random, Round Robin και PMEM-only, δεύτερος μόνο μετά την εγκατάσταση DRAM-only, η οποία είναι ιδεατή και μη κλιμακώσιμη.
- **Μνήμες Πρόσβασης**: Μείωσε σημαντικά τις πρόσβασεις σε PMEM, προτιμώντας τη χρήση

DRAM όταν ήταν κατάλληλο, διατηρώντας έτσι τη διάρκεια ζωής του PMEM και μειώνοντας την καθυστέρηση.

- **Χρήση Μνήμης**: Ο αλγόριθμος Βασισμένος σε Κατηγορίες ισορρόπησε αποτελεσματικά τη χρήση της μνήμης, προσαρμόζοντας τη στη φόρτωση και τα χαρακτηριστικά των εργασιών. Καθώς ο αριθμός των εργασιών αυξήθηκε, προτίμησε περισσότερο τη χρήση της DRAM, δείχνοντας έξυπνη διαχείριση πόρων.
- **Παράπτωμα Χρόνου Εκτέλεσης Εργασίας**: Κατέγραψε σταθερά τη μικρότερη παραλλαγή στο παράπτωμα των εργασιών, διατηρώντας ομοιόμορφη την απόδοση των εργασιών και υψηλό throughput. Αυτό υποδηλώνει καλύτερη συνέπεια στον προγραμματισμό και πιο προβλέψιμους χρόνους εκτέλεσης.

Αυτά τα αποτελέσματα επιβεβαιώνουν ότι ο συνδυασμός ταξινόμησης με τοποθέτηση με γνώμονα τους πόρους οδηγεί σε βελτιωμένη απόδοση, καλύτερη αξιοποίηση μνήμης και πιο αποδοτική χρήση υβριδικών συστημάτων μνήμης. Ο Αλγόριθμος Τοποθέτησης Βασισμένος σε Κατηγορίες όχι μόνο βελτιστοποιεί την απόδοση, αλλά συμβάλλει επίσης στην αύξηση της διάρκειας ζωής του συστήματος και στην επιχειρησιακή αποδοτικότητα, καθιστώντας τον ισχυρό υποψήφιο για πραγματική εφαρμογή σε περιβάλλοντα ετερογενών μνημών.

Συμπεράσματα και Μελλοντική Δουλειά

Η παρούσα εργασία παρουσίασε μια ολοκληρωμένη μεθοδολογία για την κατηγοριοποίηση των benchmarks με βάση τα προφίλ χρήσης μνήμης και CPU, με τον τελικό στόχο την αξιοποίηση των μοντέλων μηχανικής μάθησης για την αποδοτική κατηγοριοποίηση. Η μεθοδολογία περιλάμβανε τη συλλογή δεδομένων μέσω profiling, εξαγωγή χαρακτηριστικών, εκπαίδευση μοντέλων μηχανικής μάθησης και αξιολόγηση χρησιμοποιώντας διάφορους μετρικούς δείκτες απόδοσης. Εστιάσαμε σε τέσσερις κλάσεις benchmarks - Memory Intensive, Read Intensive, Write Intensive και CPU Intensive - με βάση τις χαρακτηριστικές χρήσεις μνήμης και CPU κατά την εκτέλεση.

Η μεθοδολογία χρησιμοποίησε επιβλεπόμενα μοντέλα μηχανικής μάθησης, όπως τα Random Forest, K-Nearest Neighbors (KNN) και Naive Bayes, τα οποία εκπαιδεύτηκαν στα δεδομένα των benchmarks, καθώς και σε μειωμένα σύνολα χαρακτηριστικών. Μέσω της ανάλυσης της σημασίας των χαρακτηριστικών, μειώσαμε τον αριθμό των χαρακτηριστικών, διατηρώντας μόνο αυτά που θεωρούνταν πιο σημαντικά για την κατηγοριοποίηση. Η Ανάλυση Κύριων Συνιστωσών (PCA) χρησιμοποιήθηκε επίσης για περαιτέρω μείωση της διάστασης του χώρου χαρακτηριστικών, επιτρέποντας μια δισδιάστατη απεικόνιση και βελτιώνοντας την απόδοση της κατηγοριοποίησης.

Η πειραματική αξιολόγηση έδειξε ότι το μοντέλο Random Forest, όταν εκπαιδεύτηκε στο μειωμένο σύνολο χαρακτηριστικών, παρουσίασε την υψηλότερη ακρίβεια (87%), κάνοντάς το το βέλτιστο μοντέλο για την κατηγοριοποίηση αυτή. Το αποτέλεσμα αυτό ανέδειξε την αποτελεσματικότητα της επιλογής χαρακτηριστικών και της μείωσης της διάστασης, καθώς και τα δύο αυτά βήματα βελτίωσαν σημαντικά την απόδοση των μοντέλων. Επίσης, η απεικόνιση μέσω PCA απέδειξε ότι οι κλάσεις των benchmarks ήταν καλά διαχωρισμένες στο μειωμένο χώρο χαρακτηριστικών, διευκολύνοντας την αποτελεσματική κατηγοριοποίηση ακόμη και με ελάχιστα χαρακτηριστικά.

Η αξιολόγηση απόδοσης μέσω των βασικών μετρικών ακρίβειας επιβεβαίωσε ότι τα μοντέλα μηχανικής μάθησης μπορούν να κατηγοριοποιήσουν με ακρίβεια τα benchmarks, προσφέροντας μια πολλά υποσχόμενη λύση για μελλοντικές εργασίες benchmarking και για ανάλυση επιδόσεων σε παρόμοια περιβάλλοντα. Αυτή η μεθοδολογική προσέγγιση μπορεί να χρησιμεύσει ως βάση για αποτελεσματική κατανομή πόρων, βελτιστοποίηση συστημάτων και ανάλυση επιδόσεων.

Παρόλο που τα αποτελέσματα της παρούσας διπλωματικής εργασίας δείχνουν υποσχόμενη απόδοση στην κατηγοριοποίηση των benchmarks, υπάρχουν αρκετές κατευθύνσεις στις οποίες αυτή η έρευνα μπορεί να επεκταθεί και να βελτιωθεί. Ορισμένες πιθανές κατευθύνσεις για μελλοντική εργασία περιλαμβάνουν:

- **Εξερεύνηση Επιπλέον Μοντέλων**: Παρά το ότι το Random Forest παρουσίασε την καλύτερη απόδοση σε αυτή την έρευνα, θα μπορούσαν να εξεταστούν και άλλα μοντέλα μηχανικής μάθησης,

όπως τεχνικές βαθιάς μάθησης ή υποστηρικτικά μηχανήματα διανυσμάτων (SVM), για περαιτέρω βελτίωση της απόδοσης. Επιπλέον, μέθοδοι συνδυασμού μοντέλων που συνδυάζουν πολλά μοντέλα μπορεί να οδηγήσουν σε καλύτερη ακρίβεια και ανθεκτικότητα.

- **Διαχείριση Μεγαλύτερων Συνόλων Δεδομένων**: Καθώς το μέγεθος του συνόλου δεδομένων αυξάνεται, η ικανότητα των μοντέλων μηχανικής μάθησης να γενικεύουν και να αποδίδουν αποτελεσματικά θα γίνει πιο σημαντική. Μια περαιτέρω έρευνα θα μπορούσε να εστιάσει στην επέκταση αυτής της μεθοδολογίας σε μεγαλύτερα και πιο περίπλοκα σύνολα δεδομένων.
- **Ενσωμάτωση Άλλων Μετρικών Profiling**: Η τρέχουσα προσέγγιση στηρίχθηκε κυρίως στην μνήμη και στην χρησιμοποίηση CPU. Μελλοντική εργασία θα μπορούσε να συμπεριλάβει επιπλέον μετρικές profiling, όπως οι αναλογίες επιτυχίας cache, καθυστέρηση ή κατανάλωση ενέργειας, που θα μπορούσαν να προσφέρουν πιο ολοκληρωμένες αναλύσεις και να βελτιώσουν την ακρίβεια της κατηγοριοποίησης.

Με την αντιμετώπιση αυτών των μελλοντικών προκλήσεων, η προσέγγιση που παρουσιάστηκε σε αυτή τη διπλωματική εργασία θα μπορούσε να εξελιχθεί σε ένα εξαιρετικά αποτελεσματικό εργαλείο για benchmarking και ανάλυση επιδόσεων σε διάφορα υπολογιστικά περιβάλλοντα, προσφέροντας τόσο προβλεπτικές δυνατότητες όσο και χρήσιμα συμπεράσματα για την βελτιστοποίηση του συστήματος.

Chapter 1

Introduction

Modern computing systems face significant performance challenges due to the memory bottleneck, a critical issue where the speed at which data is fetched from memory cannot keep up with the processing power of CPUs. As processors continue to advance, by becoming faster and more efficient, the limitations of memory access have become a major obstacle in achieving optimal performance. This bottleneck arises because DRAM, the primary volatile memory used in most systems, has not scaled in speed at the same rate as processors. As a result, applications that rely heavily on frequent memory accesses suffer from latency issues and reduced overall efficiency.

One potential solution to overcoming the memory bottleneck is to expand the memory capacity of a system using more affordable memory technologies, such as Intel Optane. Optane provides a cost-effective way to increase available memory, but its performance characteristics differ from traditional DRAM, particularly in terms of latency and bandwidth. Therefore, an efficient strategy is required to optimize the use of both DRAM and Optane memory within the same system. Proper workload classification and intelligent data allocation can help maximize performance while taking advantage of the expanded memory capacity.

To efficiently utilize a heterogeneous memory system, it is crucial to understand how different applications interact with memory resources. Applications exhibit varying behaviors, with some being more memory-intensive, and others relying heavily on computational power. The ability to classify workloads based on memory access patterns allows for better memory allocation strategies, ensuring that performance-critical applications receive the fastest available memory resources.

In this study, machine learning techniques are employed to classify applications based on their memory and computational behavior. By profiling various workloads and extracting key performance metrics, a dataset is constructed to train supervised learning models. The goal is to develop a methodology that can accurately categorize applications and provide insights into their memory demands. This classification can guide memory management strategies, improving the overall efficiency of hybrid memory systems.

The proposed approach involves profiling workloads running on DRAM and Optane memory, extracting relevant performance features, and using machine learning models to classify benchmarks into predefined categories. Feature selection and dimensionality reduction techniques are also applied to improve classification accuracy and reduce computational complexity. The trained models are evaluated across multiple scenarios to assess their effectiveness in distinguishing different workload types.

This thesis is structured as follows: Chapter 3 provides an overview of the theoretical background related to memory systems and machine learning. The proposed methodology is described in Chapter 4, detailing the profiling process, dataset creation, and machine learning models used. Chapter 5 presents the experimental results, analyzing the performance of the classification models and their implications. Finally, conclusions and potential directions for future work are discussed in Chapter 6.

Chapter 2

Related Work

The emergence of hybrid memory systems that combine high-speed DRAM with high-capacity but slower non-volatile memories such as Intel Optane Persistent Memory (PMEM) has generated considerable interest in dynamic memory management techniques. A large body of research has explored dynamic placement of data in such systems to optimize performance and resource utilization.

Most existing approaches to hybrid memory management focus on data movement and placement, either at the **object level** or the **page level**. These methods rely heavily on runtime profiling to monitor memory access behavior, and use the collected information to guide decisions about where data should reside - either in DRAM for fast access or in PMEM for capacity. Although effective in improving system performance, these strategies often incur high profiling overhead and may require frequent monitoring and migration of data, which can negate some of the performance benefits of DRAM.

On the one hand, in the category of **object placement**, the memory manager monitors the behavior of individual data structures or memory allocations to determine optimal placement within hybrid memory systems. One approach introduces a transparent and efficient mechanism for object placement in DRAM-NVM configurations, aiming to reduce write traffic to non-volatile memory and improve performance without significant changes to managed runtimes [1]. Another method proposes a performance-guided allocation policy that dynamically selects memory types based on the expected impact on performance [2]. An object-level strategy based on runtime object matching and bandwidth-aware placement has also been shown to significantly enhance performance in hybrid DRAM-PMEM setups [3]. Additionally, studies that utilize profiling tools on real hardware demonstrate that data placement optimization in heterogeneous memory environments can yield substantial gains in both performance and energy efficiency [4].

On the other hand, **page placement** approaches operate at the operating system level and rely on page-level monitoring to detect access patterns. One approach proposes a novel holistic framework for hybrid memory architectures, combining dynamic data placement with persistent heaps and periodic persistence to optimize performance in high-performance computing applications. This technique reduces redundant computations and enhances overall system efficiency [5]. Another research presents a dynamic page placement algorithm for hybrid DRAM-DCPMM systems, optimizing memory policies to improve both throughput and energy efficiency [6]. Additionally, the impact of page table placement in tiered memory systems has been studied, with a proposed method for efficiently managing page table pages by dynamically migrating them between DRAM and NVMM [7]. A further study introduces a page management scheme for multi-tiered memory systems that extends NUMA support, optimizing page placement based on both access locality and memory tier [8].

Despite the variety and maturity of these techniques, they share a common focus: optimizing *data* placement. In contrast, this work addresses a different and underexplored aspect of hybrid memory management - **process or task placement**.

According to the current understanding, no prior work has proposed a placement framework that leverages machine learning-based classification of tasks to guide their placement across memory types in a hybrid system. In the presented approach, tasks are profiled once during a controlled training phase, and then a classifier is used at runtime to predict their memory access characteristics (e.g., read-intensive, write-intensive). Based on these predictions, tasks are assigned to DRAM or PMEM, without requiring continuous monitoring or data migration.

This strategy significantly reduces runtime overhead, since there is no need to trace memory accesses continuously or to move pages and objects during execution. Furthermore, the methodology provides a lightweight and scalable approach for systems running many short-lived tasks, in which the cost of profiling and data migration may be prohibitive.

In summary, while significant research has been conducted on hybrid memory management via object and page placement, this work introduces a new, low-overhead method focused on task placement driven by machine learning classification - offering both scalability and improved resource utilization for multi-tasking systems.

Chapter 3

Theoretical Background

In this chapter, the theoretical background for the key topics explored in this thesis will be presented. The chapter begins by discussing Persistent Memory and Intel's Optane technology, which are transforming modern computing systems by enabling faster and more reliable data storage solutions. This will include an exploration of how these technologies differ from traditional memory and the advantages they offer.

Next, the chapter will delve into the Memkind API, a critical tool for memory management, particularly in systems utilizing persistent memory. The Memkind API enables efficient memory allocation and optimization, which is essential for developers to manage heterogeneous systems that include both DRAM and Optane memory.

Finally, the chapter will introduce several popular machine learning models used for training in this thesis. These models include Random Forests (RF), K-Nearest Neighbors (KNN), Naive Bayes, and Principal Component Analysis (PCA) with Support Vector Machines (SVM). A brief overview of each model's algorithmic approach, strengths, and typical applications will be provided, laying the foundation for their application in data analysis and the evaluation of memory-optimized systems.

This theoretical background will provide the necessary context to understand the integration of persistent memory technologies and machine learning algorithms, setting the stage for the practical exploration that follows.

3.1 Persistent Memory

3.1.1 Non-Volatile Memory

Persistent memory, also referred to as non-volatile or storage-class memory, represents an emerging technology that combines key attributes of both DRAM and non-volatile storage like solid-state drives (SSDs). In computer systems, there is a significant hierarchical gap between DRAM and storage disks, creating a substantial bottleneck that limits the number of memory accesses achievable per unit of time. Persistent memory aims to bridge this gap by integrating features from both memory hierarchies, offering the speed and accessibility of DRAM with the durability and data retention of traditional storage. This type of memory maintains data even when power is lost, providing the persistence of storage devices along with the higher speed, lower latency, and greater bandwidth traditionally associated with volatile memory. Persistent memory represents a significant shift in data storage, as it supports rapid access and high performance while ensuring data survival across power cycles [10]. This duality positions it between volatile DRAM, which is fast but loses data without power, and traditional non-volatile storage, which retains data but typically operates at lower speeds.

Many technologies have been developed for the fabrication of these memory types, with some distinguished by exceptional advancements, such as resistive RAM (ReRAM), Phase Change Memory

(PCM), Spin-Transfer Torque RAM(STT-RAM) and 3D XPoint [11].

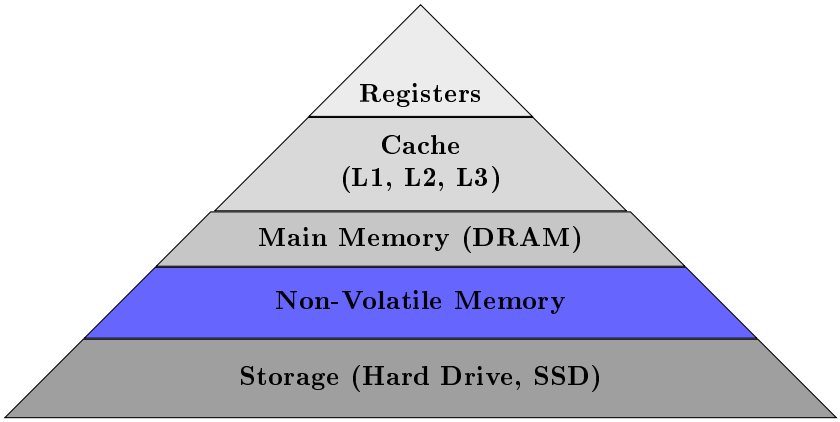


Figure 3.1.1: Memory Hierarchy Pyramid

There have been proposals for using this memory as a storage device and as an alternative to DRAM, offering potential advantages in both roles.

Given its advantageous features, such as low access latency, minimal power consumption, and high endurance cycles, compared to the existing storage devices, NVM is expected to serve as secondary storage memory alongside SSDs and hard disk drives (HDDs) [10]. The consideration of NVM as a storage device is further underscored by its inherent limitation in access endurance [11], which impacts its overall reliability and performance over time. Thus, optimizing its use is essential in order to reduce the number of read and write accesses.

However, Non-Volatile Memory can provide a scalable and power-efficient solution as a primary memory alternative to DRAM. This solution is based on the attractive characteristics of NVM, such as its higher density and nearly zero static power consumption. It also can provide a byte-addressable interface rather than a block-based accesses. While NVM technologies are offering greater capacity at comparable or lower costs than DRAM, they may exhibit lower bandwidth and access time. These characteristics of NVM could result in a significant performance difference between next-generation systems utilizing NVM and conventional systems based on DRAM for many applications [12].

3.1.2 Intel's Optane DIMM

Optane DIMM is the first scalable NVDIMM available for commercial use. Unlike current storage solutions, including Optane SSDs that use external interfaces such as PCIe, the Optane DIMM offers reduced latency, increased read bandwidth, and uses an address-based memory interface rather than a block-based NVMe interface. Compared to DRAM, it features higher density and persistence. At launch, Optane DIMM is available in three capacities: 128 GB, 256 GB and 512 GB [13].

Similar to traditional DRAM DIMMs, the Optane DC PMM is located on the memory bus and connects to the integrated memory controller (iMC) in the CPU. The Optane DC PMM is introduced alongside Intel's second-generation Xeon Scalable processors, codenamed Cascade Lake. In this setup, each CPU has two iMCs, with each iMC supporting three channels. As a result, a single CPU socket can accommodate up to six Optane DC PMMs, allowing for a maximum of 6 TB of Optane DC memory.

To ensure data persistence, the iMC operates within the asynchronous DRAM refresh (ADR) domain. Intel's ADR feature guarantees that CPU stores reaching the ADR domain will survive a power loss, meaning they will be saved to the NVDIMM within a hold-up time of less than 100 microseconds. It's important to note that the ADR domain does not encompass the processor caches, so data will only remain persistent once it has been stored in the iMC.

The iMC interacts with the Optane DC PMM through the DDR-T interface. This interface shares

a mechanical and electrical design with DDR4 but utilizes a different protocol that accommodates variable latencies, as the access latencies of Optane DC memory are not deterministic. Similar to DDR4 (which includes ECC), it features a 72-bit data bus and transfers data in cache-line (64-byte) increments for CPU load and store operations.

When a memory access request is made to the NVDIMM, it is received by the on-DIMM controller. This central controller manages most of the processing needed for the NVDIMM and facilitates access to the banks of Optane DC media.

Once an access request reaches the controller, the address undergoes internal translation. Like SSDs, the Optane DC PMM carries out internal address translation for wear leveling and bad block management. The address indirection table (AIT) translates the DIMM physical address to an internal address for the Optane DC media device. The AIT is stored in the Optane DC media, but a copy of the AIT entries is maintained in the on-DIMM DRAM.

Once the request is translated, the actual access to the storage media takes place. Since the access granularity of Optane DC media is 256 bytes, the controller translates 64-byte load/store operations into larger 256-byte accesses. This leads to write amplification, as smaller writes initiated by the CPU are processed as read-modify-write operations on the Optane DC memory by the controller.

Unlike DRAM, Optane DC memory does not require constant refreshing for data retention, which results in lower power consumption when idle. The Optane DC PMM features two configurable power budgets: the average power budget, which governs the power allowance for continuous workloads, and the peak power budget, which sets the maximum power usage during burst traffic. Both of these budgets can be adjusted by the user [9].

The performance of Intel Optane PMEM can be challenging to measure accurately due to the inherent characteristics of its persistence. Unlike traditional volatile memory, which loses data when power is removed, Optane technology retains data even without a power supply, complicating performance assessments.

Numerous studies have attempted to measure the latency and bandwidth of Intel Optane memory, often yielding varied results due to different methodologies and testing environments. While latency measurements can fluctuate significantly based on factors like system configuration, workload characteristics, and benchmarking tools, bandwidth results tend to show more consistency across different studies.

Operation	**Latency (ns)**	**Bandwidth (GB/s)**
DRAM (Read)	81	39.4
DRAM (Write)	86	13.9
Opatne (Read)	305	6.6
Optane (Write)	94	2.3

Table 3.1: DRAM and Optane Read/Write Latency and Bandwidth [9]

Some reported numbers are 305 ns for random loads, compared to 81 ns for DRAM on the same platform and 169 ns for sequential loads. Write latency is challenging to measure due to hardware limitations. A study measured it at 94 ns for Optane DC and 86 ns for DRAM. For bandwidth, it has been reported that the maximum reading bandwidth is 6.6 GB/s, whereas the writing bandwidth cannot surpass 2.3 GB/s [9].

3.2 Memkind API

Memkind, an open-source library developed by Intel, provides a flexible and efficient memory allocation framework designed for applications that utilize diverse memory types, such as volatile and persistent

memory. This library allows developers to allocate and control memory from various sources, catering to the unique requirements of their applications.

The library offers APIs that facilitate memory allocation from sources such as DRAM (volatile memory), Intel Optane DC Persistent Memory (persistent memory), and other advanced memory technologies. Applications can either specify the desired memory type or rely on Memkind to automatically select the most suitable source based on predefined criteria.

Memkind is designed for high performance and minimal overhead, utilizing advanced memory allocation algorithms based on jemalloc along with specific optimizations to minimize latency and maximize throughput. By harnessing the distinct features of various memory types, such as the low-latency access of persistent memory, it boosts the overall performance of applications.

Memkind proves especially advantageous for applications that benefit from multiple memory types, including in-memory databases, big data analytics, high-performance computing, and other memory-intensive tasks. By leveraging Memkind, developers can efficiently manage memory resources, enhance performance, and take full advantage of the distinct capabilities offered by different memory technologies.

Memkind is designed to work seamlessly with existing memory management APIs and libraries, supporting standard memory allocation functions such as malloc() and free(). This compatibility allows for straightforward integration with minimal code modifications [14].

The ISO C programming language standard offers a user-level interface commonly used for memory management in numerous applications, either directly or indirectly. This interface includes the familiar set of APIs:

- `void *malloc(size_t size):` The malloc function allocates space for an object whose size is specified by size and whose representation is indeterminate [15].
- `void *calloc(size_t nmemb, size_t size):` The calloc function allocates space for an array of nmemb objects, each of whose size is size. The space is initialized to all bits zero [15].
- `void *realloc(void *ptr, size_t size):` The realloc function deallocates the old object pointed to by ptr and returns a pointer to a new object that has the size specified by size. The contents of the new object shall be the same as that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have unspecified values [15].
- `void free(void *ptr):` The free function causes the space pointed to by ptr to be deallocated, that is, made available for further allocation. If ptr is a null pointer, no action occurs. Otherwise, if the argument does not match a pointer earlier returned by a memory management function, or if the space has been deallocated by a call to free or realloc, the behavior is undefined [15].

The memkind library adapts these APIs by prefixing their names with "memkind_" and extending the interface to include an additional argument: the "kind" of memory. The specifics of what this argument represents will be covered later, but this design allows for a plug-in architecture that can evolve alongside advancements in hardware and policy features. Notably, the "kind" of memory dictates both the selection of hardware and the application of policies to the allocated memory [16].

The functions outlined in this section define a heap manager with an interface based on the ISO C standard APIs. However, unlike the standard APIs, the user is required to specify the type of memory as the first argument in each function.

- `void *memkind_malloc(memkind_t kind, size_t size):` Allocates size bytes of uninitialized memory of the specified kind. The allocated space is suitably aligned (after possible pointer coercion) for storage of any type of object. If size is 0, then memkind_malloc() returns NULL [17].
- `void *memkind_calloc(memkind_t kind, size_t num, size_t size):` Allocates space for num objects each size bytes in length in memory of the specified kind. The result is identical to

calling memkind_malloc() with an argument of num * size, with the exception that the allocated memory is explicitly initialized to zero bytes. If num or size is 0, then memkind_calloc() returns NULL [17].

- `void *memkind_realloc(memkind_t kind, void *ptr, size_t size):` Changes the size of the previously allocated memory referenced by ptr to size bytes of the specified kind. The contents of the memory remain unchanged up to the lesser of the new and old sizes. If the new size is larger, the contents of the newly allocated portion of the memory are undefined. Upon success, the memory referenced by ptr is freed and a pointer to the newly allocated memory is returned [17].
- `void memkind_free(memkind_t kind, void *ptr):` Causes the allocated memory referenced by ptr to be made available for future allocations. This pointer must have been returned by a previous call to memkind_malloc(), memkind_calloc() or memkind_realloc(). Otherwise, if memkind_free(*kind*, *ptr*) has already been called before, undefined behavior occurs. If ptr is NULL, no operation is performed. In cases where the kind is unknown in the context of the call to memkind_free() NULL can be given as the kind, but this will require an internal lookup for correct kind [17].

It also employs the following functions to initialize and manage the necessary configurations for handling the different types of memory.

- `int memkind_create_kind(memkind_memtype_t memtype_flags, memkind_policy_t policy, memkind_bits_t flags, memkind_t *kind):` Creates kind that allocates memory with specific memory type, memory binding policy and flags (see MEMORY FLAGS section). The memtype_flags (see MEMORY TYPES section) determine memory types to allocate, policy argument is policy for specifying page binding to memory types selected by memtype_flags. Returns zero if the specified kind is created successfully or an error code from the ERRORS section if not [17].
- `int memkind_destroy_kind(memkind_t kind):` Destroys previously created kind object, which must have been returned by a previous call to memkind_create_kind() [17].

3.3 Machine Learning Models

This section discusses the machine learning models used in the experiments. The models presented are designed for classification tasks and are among the most commonly used in the field of machine learning. Before introducing the models, the evaluation metrics used to measure their accuracy and performance will be presented to provide a clear understanding of how their effectiveness is assessed.

3.3.1 Accuracy Metric for Machine Learning Models

Evaluating the performance of the machine learning models in this study is based solely on the **Accuracy** metric. Accuracy is defined as the ratio of correctly classified instances to the total number of instances, and is calculated as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3.1}$$

In this equation, TP and TN represent the true positives and true negatives, while FP and FN denote the false positives and false negatives, respectively. Accuracy provides a straightforward and intuitive measure of model performance, which is particularly suitable for balanced datasets.

3.3.2 Decision Tree

The Decision Tree model is a versatile classification approach that learns to categorize objects through inductive analysis of a dataset with known classifications.

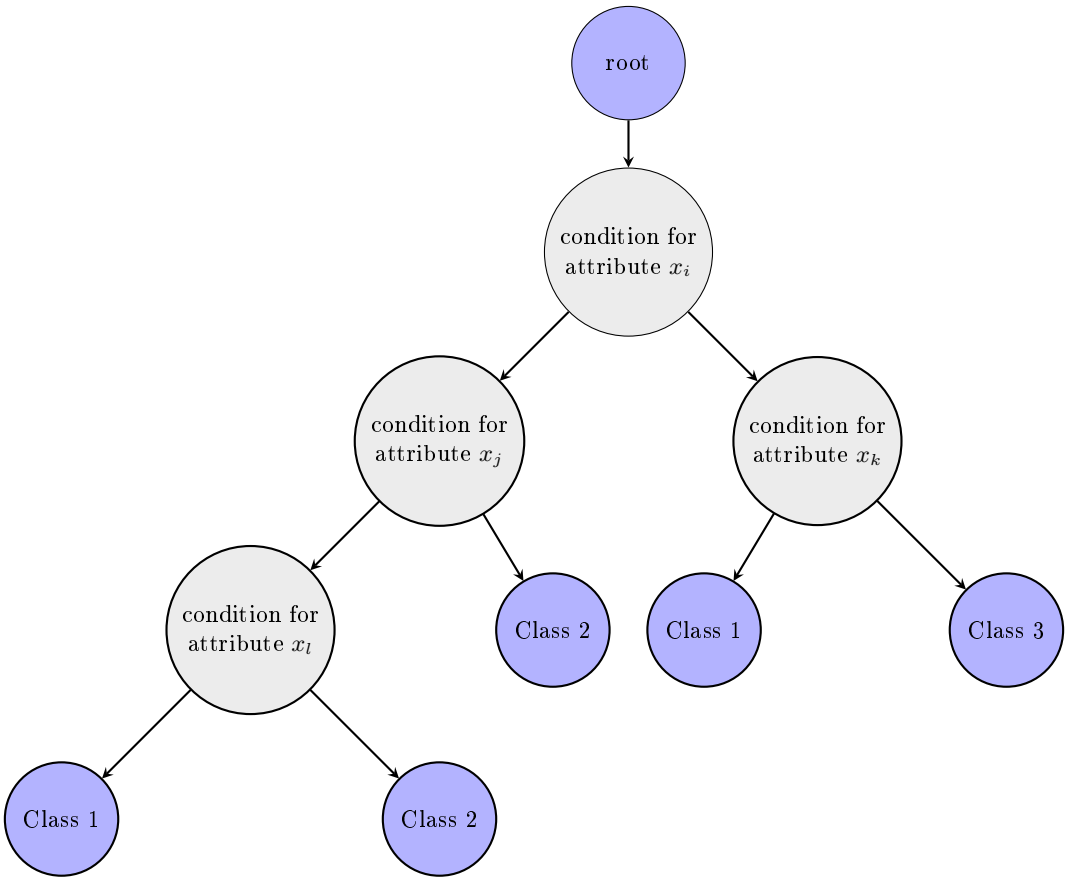


Figure 3.3.1: Decision Tree Model

Each object is described by a set of attributes which capture essential characteristics. When these attributes are sufficiently informative, a decision tree can be constructed to accurately classify all objects in the training set, often with multiple possible correct trees. However, the true objective of induction is to extend beyond the training data, enabling the model to correctly classify previously unseen objects. Achieving this requires the decision tree to identify meaningful patterns and relationships between an object's class and its attribute values [18].

A decision tree is composed of test nodes, or attribute nodes, which are connected to two or more subtrees, and leaf nodes, or decision nodes, which are labeled with a class representing the final decision. Each test node evaluates a specific attribute of an instance, with the resulting outcome determining the path to one of the subtrees.

The process of obtaining a solution using decision trees begins with the preparation of a data set containing previously solved cases. This data set is then divided into two subsets: a training set, used to induct the decision tree, and a testing set, which is used to evaluate the performance of the model. The training set helps in building the decision tree, while the testing set helps to assess its key metrics. When making a decision for an unsolved case, the process starts at the root node of the decision tree (see Figure 3.3.1). Moving through the attribute nodes (purple nodes in Figure 3.3.1), the branches are selected based on the matching attribute values of the unsolved case, continuing until the leaf node is reached (green nodes in Figure 3.3.1), representing the final decision [19].

3.3.3 Random Forest

Random Forest is an advanced ensemble learning method built upon decision trees. It enhances predictive accuracy, mitigates overfitting, and improves overall model performance by combining the outputs of multiple trees. By aggregating predictions, Random Forest creates a more robust and reliable framework, making it widely applicable in machine learning tasks.

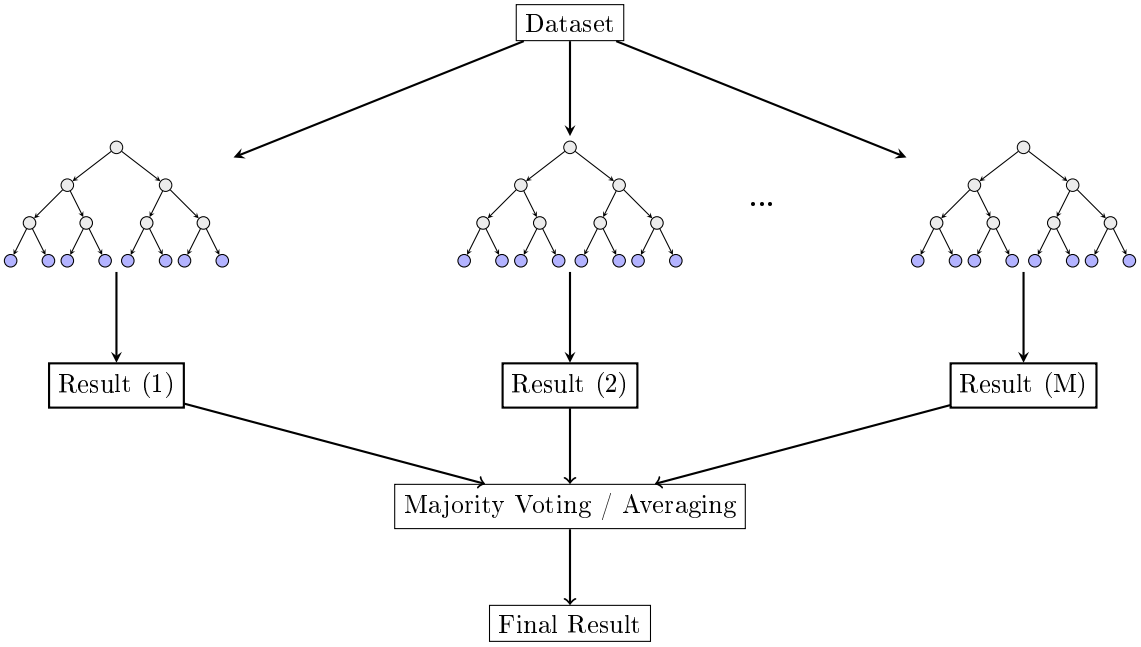


Figure 3.3.2: Random Forest Model

A Random Forest consists of multiple decision tree classifiers, where each tree independently predicts the class of a given input, and the final classification is determined through a majority vote among all trees [20].

To construct a Random Forest, M decision trees are trained as base learners. Each tree is built using a distinct bootstrap sample, generated by randomly selecting data points from the training set with replacement. This process introduces diversity among the trees and helps prevent overfitting. Furthermore, at each node, a random subset of features is chosen for splitting rather than using all available features, further enhancing tree diversity [21].

After training, the individual tree predictions are aggregated - through majority voting for classification and averaging for regression - resulting in a more accurate and stable model compared to a single decision tree. This ensemble strategy improves generalization and makes Random Forest a powerful and versatile machine learning algorithm used in various domains.

The performance of the Random Forest model is largely determined by several key parameters:

- **Number of trees M**: The total number of decision trees in the ensemble.
- **Splitting directions per node *mtry***: The number of possible feature directions considered when splitting a node in each tree.
- **Node size**: The minimum number of samples required in a cell before it is no longer split.

According to [22] optimal parameter selection enhances model accuracy. A higher number of trees (M) generally leads to more accurate predictions without the risk of overfitting. For the number of splitting directions, a practical approach is to maximize the number of features considered at each split, constrained only by computational resources. As for node size, a value of 1 is recommended for classification tasks, while 5 is preferred for regression.

Random Forest can be effectively used to identify the most important variables contributing to classification tasks. By analyzing the decrease in impurity or permutation-based importance, the model ranks features based on their impact on prediction accuracy. This capability is particularly valuable in domains where understanding key predictors is essential. Several studies [23] have focused on leveraging Random Forest for feature selection, aiming to extract the most influential variables that drive classification decisions. These approaches help improve model interpretability, reduce dimensionality, and enhance computational efficiency while maintaining predictive performance.

3.3.4 K-Nearest Neighbors (KNN)

The K-Nearest Neighbor (KNN) algorithm is a widely utilized classification method in statistical pattern recognition. In this approach, each class is represented by a set of sample prototypes, which serve as the training dataset of pattern vectors for that class. When classifying an unknown vector, the algorithm identifies its k nearest neighbors from the prototype set and assigns a class label based on the majority vote. To minimize ties in overlapping class regions, k is typically chosen as an odd number. Despite its simplicity and intuitive nature, KNN demonstrates a low error rate in practical applications [24].

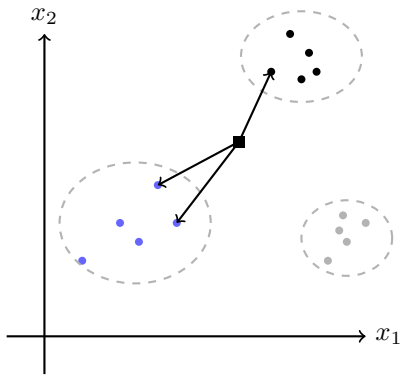


Figure 3.3.3: K-Nearest Neighbors Model

In the example shown in Figure 3.3.3, the K-Nearest Neighbors (KNN) model is applied in a two-dimensional feature space defined by attributes x_1 and x_2. An unknown data point needs to be classified, and the algorithm identifies its three nearest neighbors ($k = 3$). Among these neighbors, two belong to the blue class, while one belongs to the black class. Since KNN follows a majority voting rule, the unknown point is assigned to the blue class. This demonstrates how KNN classifies new data based on local patterns, with the choice of k influencing the final decision.

The performance of the K-Nearest Neighbors (KNN) algorithm depends on several key parameters that influence classification accuracy and generalization. These parameters include:

- **Number of neighbors (k)**: Determines how many nearest neighbors are considered when classifying a new data point.
- **Distance metric**: The choice of distance function, such as Euclidean, Manhattan, or Minkowski distance, affects how similarity between points is measured.
- **Weighting scheme**: Neighbors can be weighted equally (uniform weighting) or assigned weights based on their distance (inverse distance weighting), influencing the classification decision.

To optimize these parameters, cross-validation is an effective technique. By partitioning the dataset into multiple folds and evaluating different parameter configurations on training and validation sets, cross-validation helps determine the optimal values, improving model generalization and minimizing overfitting [25].

In the KNN algorithm, having a large number of features can significantly increase computational complexity, as the distance between data points must be computed across all dimensions. Since KNN relies on similarity measures, the presence of many features leads to higher-dimensional space, making the distance calculations more expensive and potentially slowing down the model. Additionally, if a significant portion of these features is irrelevant or redundant, they can negatively impact classification accuracy. This is because irrelevant features introduce noise, reducing the effectiveness of the distance metric in distinguishing between classes [26].

To overcome these challenges, this thesis proposes two approaches to reduce the number of features:

- **Principal Component Analysis (PCA)** [26]: This method transforms the feature space into a lower-dimensional representation while preserving essential information, as discussed in Section 3.3.7.
- **Feature Selection using Random Forest**: The most important features are selected based on feature importance scores derived from a Random Forest model, ensuring that only the most relevant information is retained.

By reducing the feature set through these methods, the computational cost is minimized, and the model's performance is enhanced by focusing on the most relevant information.

3.3.5 Naive Bayes

Naive Bayes classifiers are supervised machine learning models used for classification. They work based on Bayes' Theorem, which helps calculate the probability of a class given certain features. The key idea is to use these probabilities to assign data to the most likely category.

Bayes' Theorem states that the probability of a hypothesis A given some B is calculated by dividing the overall probability of both the hypothesis and the data occurring together by the overall probability of the data alone [27].

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{3.3.2}$$

Naive Bayes is a simple and fast algorithm that is easy to implement [28]. It does not require a lot of training data to perform well, making it useful when there is limited information available [29]. Another key advantage is that it is very fast computationally and it works efficiently with high-dimensional data, but this is true only if certain assumptions hold [30].

These assumptions are:

- The features are independent of each other.
- The data follows a specific probability distribution, often Gaussian or multinomial [31].

When the first assumption is met, Naive Bayes can handle multi-feature classification by using the following formula:

$$P(Y \mid X_1, X_2, \ldots, X_n) = \frac{P(Y) \prod_{i=1}^{n} P(X_i \mid Y)}{P(X_1, X_2, \ldots, X_n)} \tag{3.3.3}$$

Where:

- $P(Y \mid X_1, X_2, \ldots, X_n)$ is the probability of class Y given the features $X_1, X_2, \ldots, X_n$.
- $P(Y)$ is the prior probability of the class.
- $P(X_i \mid Y)$ is the probability of feature X_i given the class Y.
- $P(X_1, X_2, \ldots, X_n)$ is the probability of the features.

Assuming the features are independent simplifies the joint probability of the features to the product of individual probabilities, thereby making the computation more efficient.

Even though the assumptions made are not always completely true in real-life data, the algorithm often performs well in many real-world applications [31].

3.3.6 Support Vector Machine (SVM)

The Support Vector Machine (SVM) is a supervised learning algorithm commonly applied to both classification and regression problems. Although it can be used for regression, SVM is particularly effective for classification tasks.

The primary goal of SVM is to identify the best hyperplane in an N-dimensional space that effectively divides data points into distinct classes. The algorithm works by maximizing the margin between the nearest data points of each class.

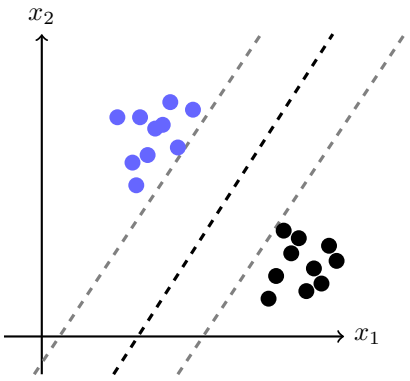


Figure 3.3.4: Support Vector Machine Model

SVM uses either a linear or nonlinear separation surface depending on the complexity of the data [32]. For linearly separable data, a straight line or hyperplane is used to separate different classes. However, for more complex data that are not linearly separable, SVM applies various kernels to transform the data into a higher-dimensional space. This allows the algorithm to find a linear separation in the transformed space, even if the data are not linearly separable in the original space. Kernels play a crucial role in enabling SVM to handle more complex classification tasks. The following kernel functions are commonly used [33]:

- **Linear Kernel**: $K(x, x') = x^T x$
- **Polynomial Kernel**: $K(x, x') = (x^T x' + 1)^d$
- **Radial Basis Function (RBF) Kernel**: $K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)$

SVM tends to work pretty well as the number of features increases and can handle high-dimensional data effectively [34]. However, when the number of classes goes beyond two, it becomes more difficult to create an accurate model. This is because SVM is designed for binary classification, and extending it to handle multiple classes requires extra methods like "one-vs-one" or "one-vs-rest" [35].

3.3.7 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a common method used to make large datasets simpler by reducing the number of variables while keeping most of the important information. The main purpose of PCA is to make complex data easier to understand and analyze. This is especially helpful when working with datasets that have many variables.

PCA works by transforming the original features into a new set of variables called principal components. These principal components are linear combinations of the original features and are ordered in terms

of the variance they explain. The first principal component captures the highest variance in the data, the second captures the second-highest variance, and so on. By selecting only the first few principal components, it is possible to reduce the number of dimensions while still keeping most of the important information.

Principal Component Analysis (PCA) offers several benefits when working with large and complex datasets. Some of its main advantages include:

- **Reduces Complexity**: PCA decreases the number of variables in a dataset while keeping most of the important information. This makes data easier to analyze [36].
- **Improves Visualization** : By reducing data to two or three dimensions, PCA helps in creating visual representations, making it easier to understand patterns in the data [37].
- **Speeds Up Computation**: A dataset with fewer variables requires less computational power and runs faster in analysis and machine learning processes [36].

PCA is especially useful when dealing with datasets that have many variables, as it reduces the complexity of the data without losing too much important information. It is commonly used in fields such as image processing, material science, and finance , where the data is often high-dimensional [36], [38], [39].

Chapter 4

Proposed Methodology

4.1 Overview

This chapter describes the methodology used in this research to classify benchmarks using machine learning techniques and evaluate their classification using a multitask algorithm. The methodology consists of several key stages, including data collection, feature extraction, classification, predictive modeling, and evaluation. Each stage plays an important role in ensuring that the classification is accurate and meaningful.

The first step in the methodology is the **profiling and characterisation** of benchmarks, where each benchmark is analyzed to extract relevant features. These features describe the characteristics of the benchmark, such as performance metrics, computational behavior, and resource usage. Once the benchmarks are profiled, the next step is **classification**. This involves grouping benchmarks into categories based on their extracted features. After classification, the research applies **predictive modeling** to build machine learning models that can automatically determine the class of a new benchmark. Several machine learning models are tested, including random forests, k-nearest neighbors, and naive bayes models. The goal is to find the most effective model for classification based on its accuracy. To further assess the effectiveness of the classification results, a **class-based placement algorithm** is introduced. This algorithm assigns incoming tasks to memory types based on their predicted class - determined by the machine learning model - and the current state of the system. The goal is to intelligently distribute tasks between DRAM and PMEM in a way that leverages each memory's strengths. The performance of this approach is compared against several baseline placement algorithms to evaluate its efficiency and impact on overall system performance.

The following sections provide a detailed explanation of each stage, outlining the specific techniques and methodologies used to achieve reliable and accurate benchmark classification.

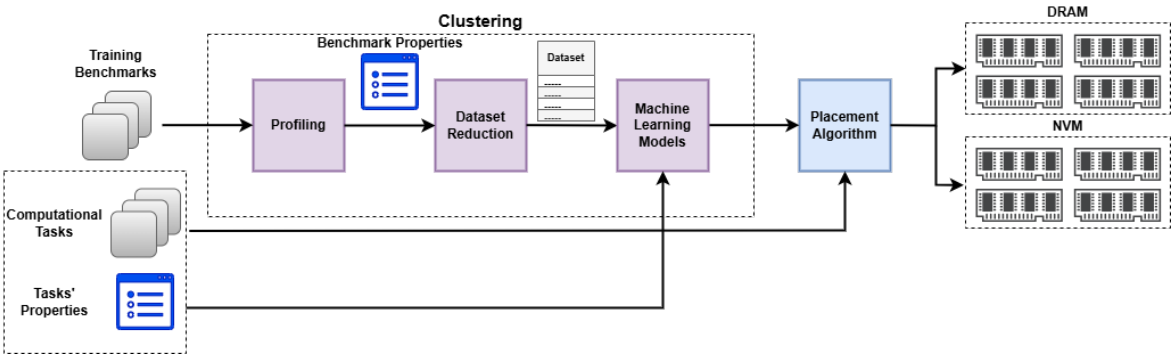


Figure 4.1.1: Overview of the proposed methodology.

4.2 Profiling and Characterisation

Profiling and characterisation is the foundational stage in the methodology, where each benchmark is thoroughly analyzed to extract the relevant features that will be used for classification. The primary objective of this stage is to create a dataset that captures a wide variety of benchmark behaviors, ensuring that the resulting data reflects diverse performance characteristics and computational patterns.

To build a dataset that encompasses a range of behaviors, a benchmark suite was developed by combining programs from existing benchmark suites, including GAP, Parsec, Rodinia, and EEMBC. These suites are widely used to evaluate how well systems perform in various scenarios.

- **Parsec**: This suite is used to test parallel applications on modern multicore processors [40].
- **Rodinia**: A benchmark suite designed to test heterogeneous systems for parallel programs [41].
- **EEMBC**: Benchmarks that focus on evaluating the performance of embedded systems [42].
- **GAP**: The GAP suite is used for testing graph-based programs [43].

By selecting programs from multiple sources, the dataset captures a broad spectrum of computational tasks, from scientific computing and parallel applications to embedded system benchmarks. In addition to combining benchmarks from different suites, a variety of input configurations were introduced to increase the number of tasks and amplify the diversity of behaviors. These variations were designed to challenge the classifiers by incorporating a wide range of resource usage patterns, execution times, and computational demands. This approach ensures that the resulting dataset is rich in behaviors, making it suitable for evaluating the effectiveness of machine learning models in classifying different types of benchmarks.

Benchmark	Input	Description
backprop	Elements: 30,000,000	Back propagation on NN
cfd	-	Fluid dynamics simulation
heartwall	Frames: 104, Threads: 2	Heart wall shape tracking
hotspot	Rows/Cols: 1024, Time: 100,000, Threads: 2	Thermal simulation
hotspot3D	Rows/Cols: 512, Layers: 8, Iterations: 5000	Thermal simulation in 3D
lavaMD	Cores: 2, Boxes: 35	Particle simulation
leukocyte	Frames: 100, Threads: 2	Leukocyte tracking
lud	Matrix size: 15000	LU decomposition
myocyte	X: 10,000, Workload: 200, Mode: 1, Threads: 2	Cardiac myocyte simulation
particlefilter	X: 4096, Y: 4096, Z: 100, No. of particles: 10000	Target location estimation
pathfinder	Width:100,000, No. of steps: 5,000	Optimal path search
streamcluster	Min k:10, Max k: 100, Dimensions: 256, N: 65,536, Chunk size: 65,536, Cluster size:1,000, Threads: 4	Clustering task

Table 4.1: Rodinia Benchmarks

Benchmark	Input	Description
blackscholes	native	Solves the Black-Scholes differential equation
bodytrack	native	Tracks the 3D pose of a human body through an image sequence
canneal	native	Simulates annealing (SA) of a chip design
dedup	native	Compresses a data stream with a combination called 'deduplication'

Table 4.2: Parsec Benchmarks

Benchmark	Input	Description
audiomark	-	Machine learning performance benchmark
coremark	-	Measures the performance of CPUs and embedded microcrontrollers
coremark_pro	-	A multi-processor coremark benchmark
securemark_tls	-	Cryptography benchmark
securemark_v2	-	Cryptography benchmark Version 2

Table 4.3: EEMBC Benchmarks

Benchmark	Input	Description
bfs_150	Iterations: 150 , Vertices: 10^{20}	Breadth-First Search with 150 iterations
bfs_200	Iterations: 200 , Vertices: 10^{20}	Breadth-First Search with 200 iterations
bfs_300	Iterations: 300 , Vertices: 10^{20}	Breadth-First Search with 300 iterations
bc_50	Iterations: 50 , Vertices: 10^{20}	Betweenness Centrality with 50 iterations
bc_70	Iterations: 70 , Vertices: 10^{20}	Betweenness Centrality with 70 iterations
bc_100	Iterations: 100 , Vertices: 10^{20}	Betweenness Centrality with 100 iterations
cc_150	Iterations: 150 , Vertices: 10^{20}	Connected Components (CC) 150 times
cc_200	Iterations: 200 , Vertices: 10^{20}	Connected Components (CC) 200 times
cc_300	Iterations: 300 , Vertices: 10^{20}	Connected Components (CC) 300 times
cc_sv_150	Iterations: 150 , Vertices: 10^{20}	CC using the Shiloach-Vishkin 150 times [44]
cc_sv_200	Iterations: 200 , Vertices: 10^{20}	CC using the Shiloach-Vishkin 200 times
cc_sv_300	Iterations: 300 , Vertices: 10^{20}	CC using the Shiloach-Vishkin 300 times
pr_150	Iterations: 150 , Vertices: 10^{20}	Page-Rank (PR) with 150 iterations
pr_200	Iterations: 200 , Vertices: 10^{20}	Page-Rank (PR) with 200 iterations
pr_300	Iterations: 300 , Vertices: 10^{20}	Page-Rank (PR) with 300 iterations
pr_spmv_75	Iterations: 75 , Vertices: 10^{20}	PR using sparse matrices with 75 iterations
pr_spmv_100	Iterations: 100 , Vertices: 10^{20}	PR using sparse matrices with 100 iterations
pr_spmv_150	Iterations: 150 , Vertices: 10^{20}	PR using sparse matrices with 150 iterations
tc_75	Iterations: 75 , Vertices: 10^{20}	Triangle Counting with 75 iterations
tc_100	Iterations: 100 , Vertices: 10^{20}	Triangle Counting with 100 iterations
tc_150	Iterations: 150 , Vertices: 10^{20}	Triangle Counting with 150 iterations
sssp_75	Iterations: 75 , Vertices: 10^{20}	Single-Source Shortest Paths with 75 iterations
sssp_100	Iterations: 100 , Vertices: 10^{20}	Single-Source Shortest Paths with 100 iterations
sssp_150	Iterations: 150 , Vertices: 10^{20}	Single-Source Shortest Paths with 150 iterations

Table 4.4: GAP Benchmarks

The benchmark suite was executed two times: once using the DRAM memory chips and once using only the sockets mounted by Intel's Optane DIMMs. This setup enabled a comparative analysis of benchmark performance under different memory architectures. Execution time was recorded for each run, while Intel's Performance Counter Monitor (PCM) was employed to collect detailed performance metrics, offering insights into resource utilization. For each run, the following metrics were measured separately:

- **Execution time:** The total time taken for the benchmark to complete, indicating the overall performance efficiency for each memory configuration.
- **Instructions per Cycle (IPC) over time:** A measure of CPU utilization, showing the number of instructions executed per clock cycle throughout the execution.
- **L3 Cache Hit Ratio over time:** The proportion of cache accesses that result in a cache hit in the L3 cache, providing insight into how effectively the L3 cache reduces memory latency.

- **L3 Cache Misses over time:** The number of instances when data was not found in the L3 cache.
- **Memory Read Throughput over time:** The rate at which data is read from memory, indicating the efficiency of memory read operations throughout execution.
- **Memory Write Throughput over time:** The rate at which data is written to memory, showing the efficiency of memory write operations during benchmark execution.
- **Total Memory Read Accesses:** The total number of memory read accesses during execution on Optane, highlighting how much the benchmark interacts with the memory. This metric is not applicable for DRAM, only for Optane.
- **Total Memory Write Accesses:** The total number of memory write accesses on Optane, indicating the level of write activity in the benchmark when using persistent memory. This metric is not applicable for DRAM, only for Optane.

These performance metrics provide valuable insights into how the benchmarks perform under varying memory architectures, allowing for a comprehensive understanding of their resource usage and computational behavior. The extracted features are essential for building accurate machine learning models capable of classifying benchmarks based on their performance characteristics.

Metric	For DRAM	For Optane
Execution time	Yes	Yes
Instructions per Cycle (IPC) over time	Yes	Yes
L3 Cache Hit Ratio over time	Yes	Yes
L3 Cache Misses over time	Yes	Yes
Memory Read Throughput over time	Yes	Yes
Memory Write Throughput over time	Yes	Yes
Total Memory Read Accesses	No	Yes
Total Memory Write Accesses	No	Yes

Table 4.5: Metrics measured during benchmark runs on DRAM and Optane.

4.3 Profiling Results

This section presents the profiling results obtained from executing the benchmark suite on both DRAM and Intel Optane memory. Each benchmark execution was analyzed based on multiple performance metrics, and the results are visualized to highlight differences in execution behavior.

4.3.1 Execution Time

Figure 4.3.1 shows the execution time for each benchmark when running on DRAM and Optane. The difference in execution times reflects the impact of memory technology on performance.

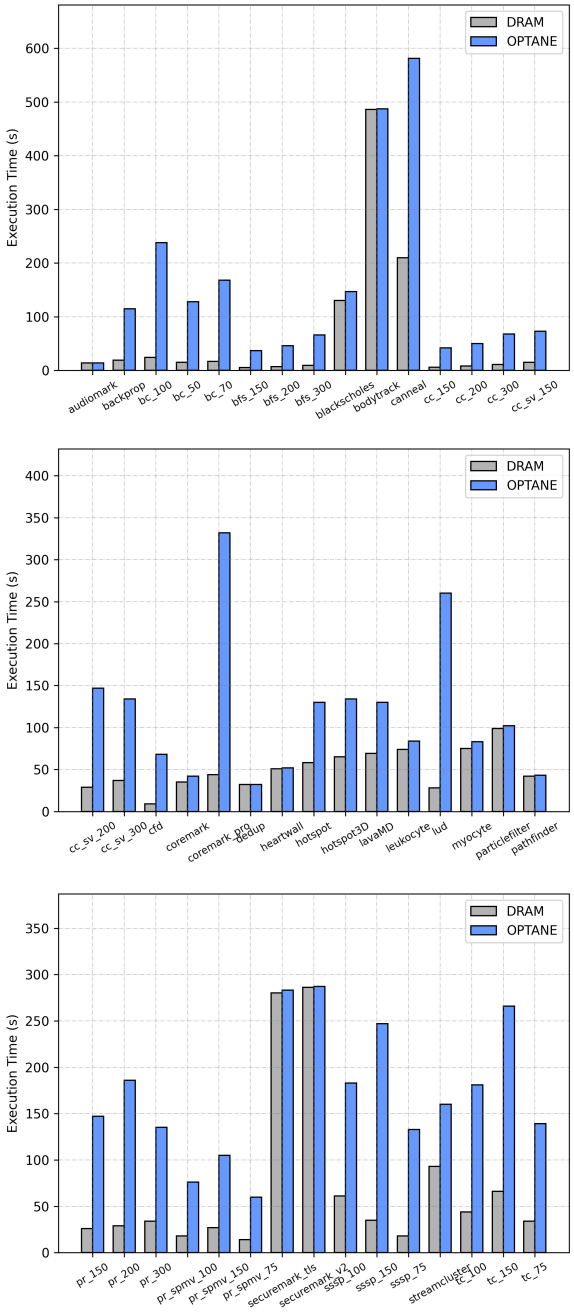


Figure 4.3.1: Execution times of benchmarks on DRAM vs. Optane.

4.3.2 Instructions Per Cycle (IPC)

Figure 4.3.2 presents the IPC values recorded during benchmark execution. IPC variations indicate different computational intensities of the benchmarks.

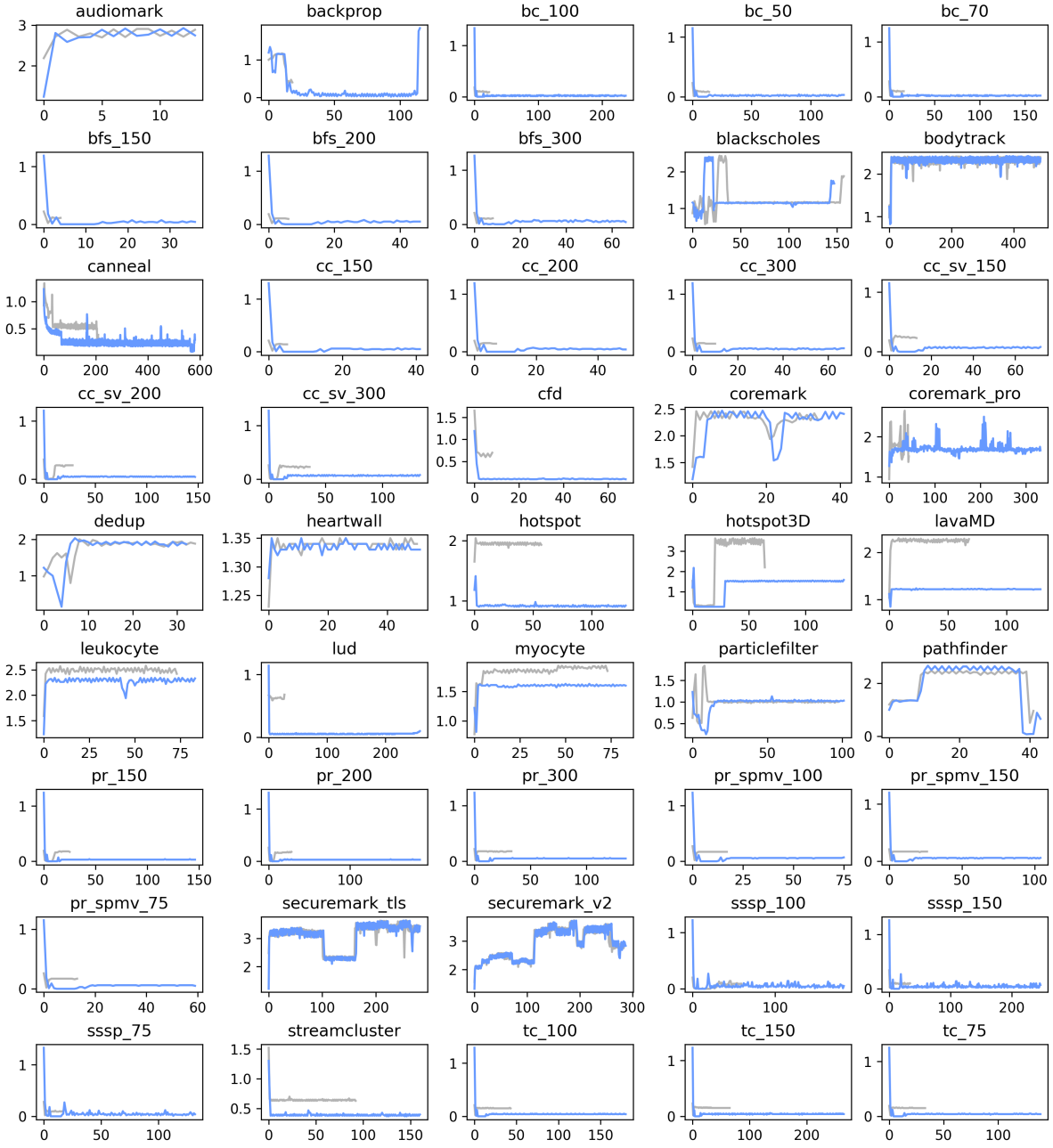


Figure 4.3.2: IPC of benchmarks on DRAM (gray) vs. Optane (blue).

4.3.3 L3 Cache Performance

Figures 4.3.3 and 4.3.4 depict the L3 cache hit ratio and L3 cache misses per second, respectively. These metrics illustrate how well benchmarks utilize the cache.

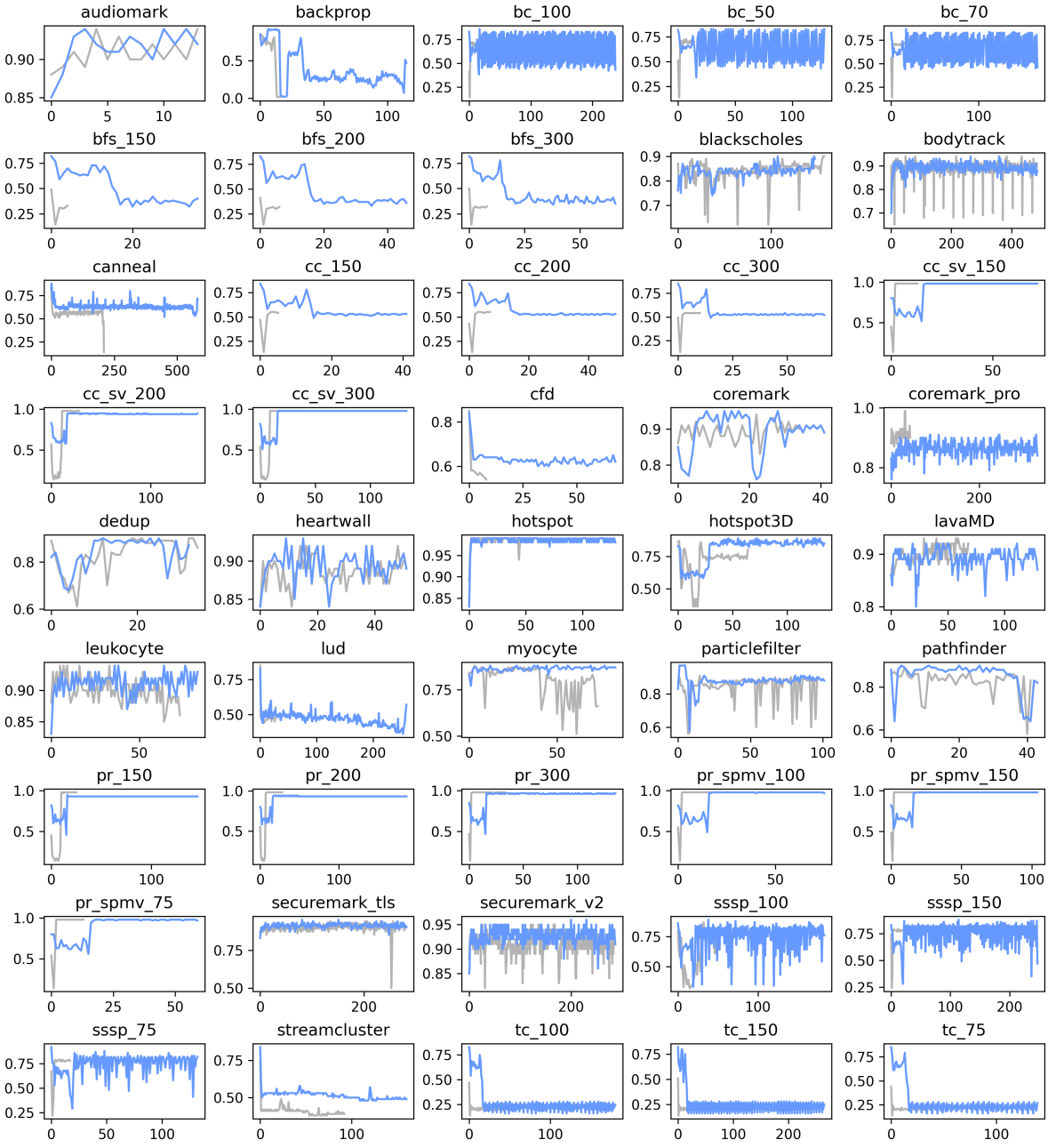


Figure 4.3.3: L3 cache hit ratio of benchmarks on DRAM (gray) vs. Optane (blue).

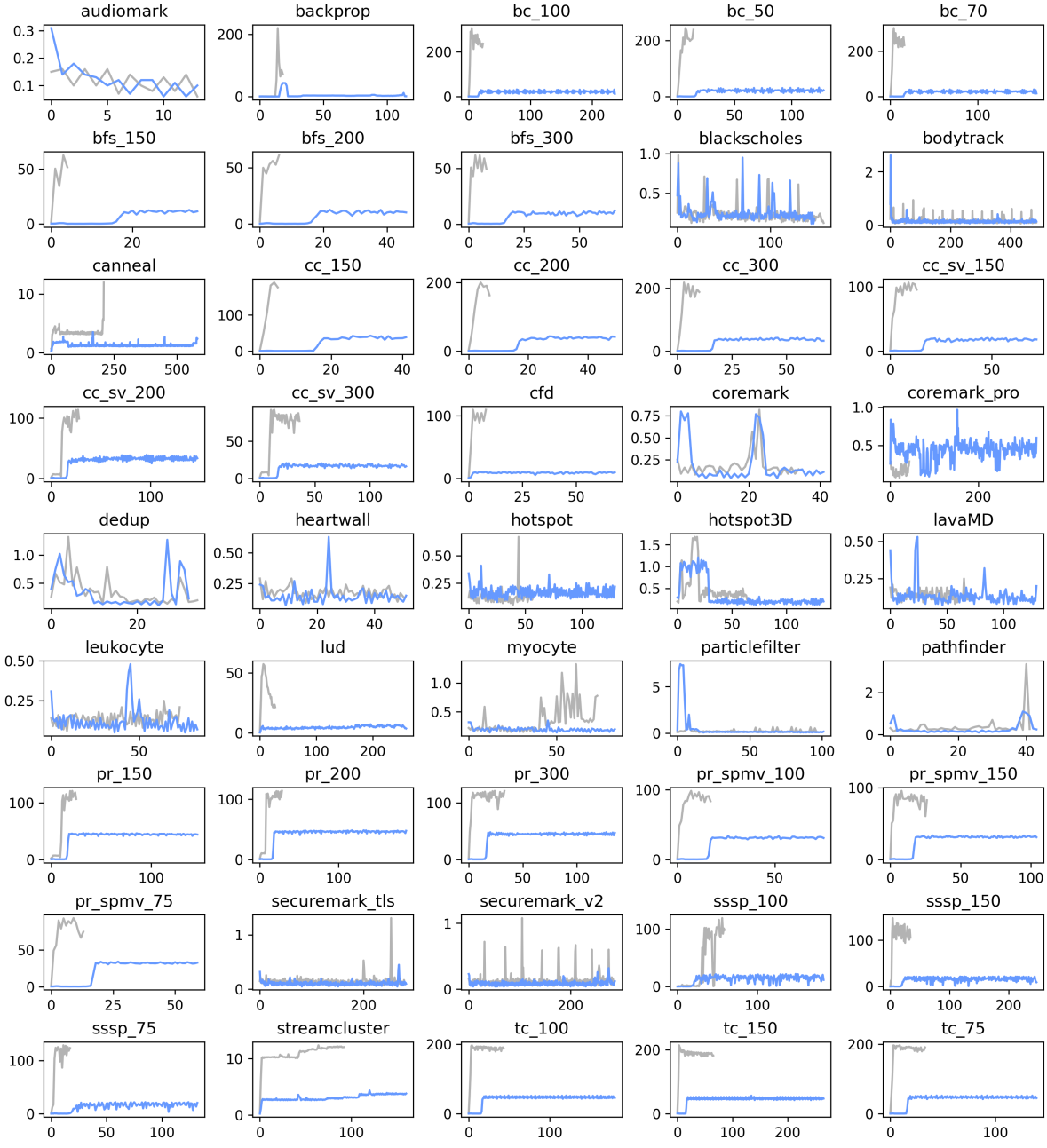


Figure 4.3.4: L3 cache misses (KB) per second of benchmarks on DRAM (gray) vs. Optane (blue).

4.3.4 Memory Bandwidth

Figures 4.3.5 and 4.3.6 illustrate the read and write bandwidths measured during execution. These metrics indicate how memory-intensive each benchmark is.

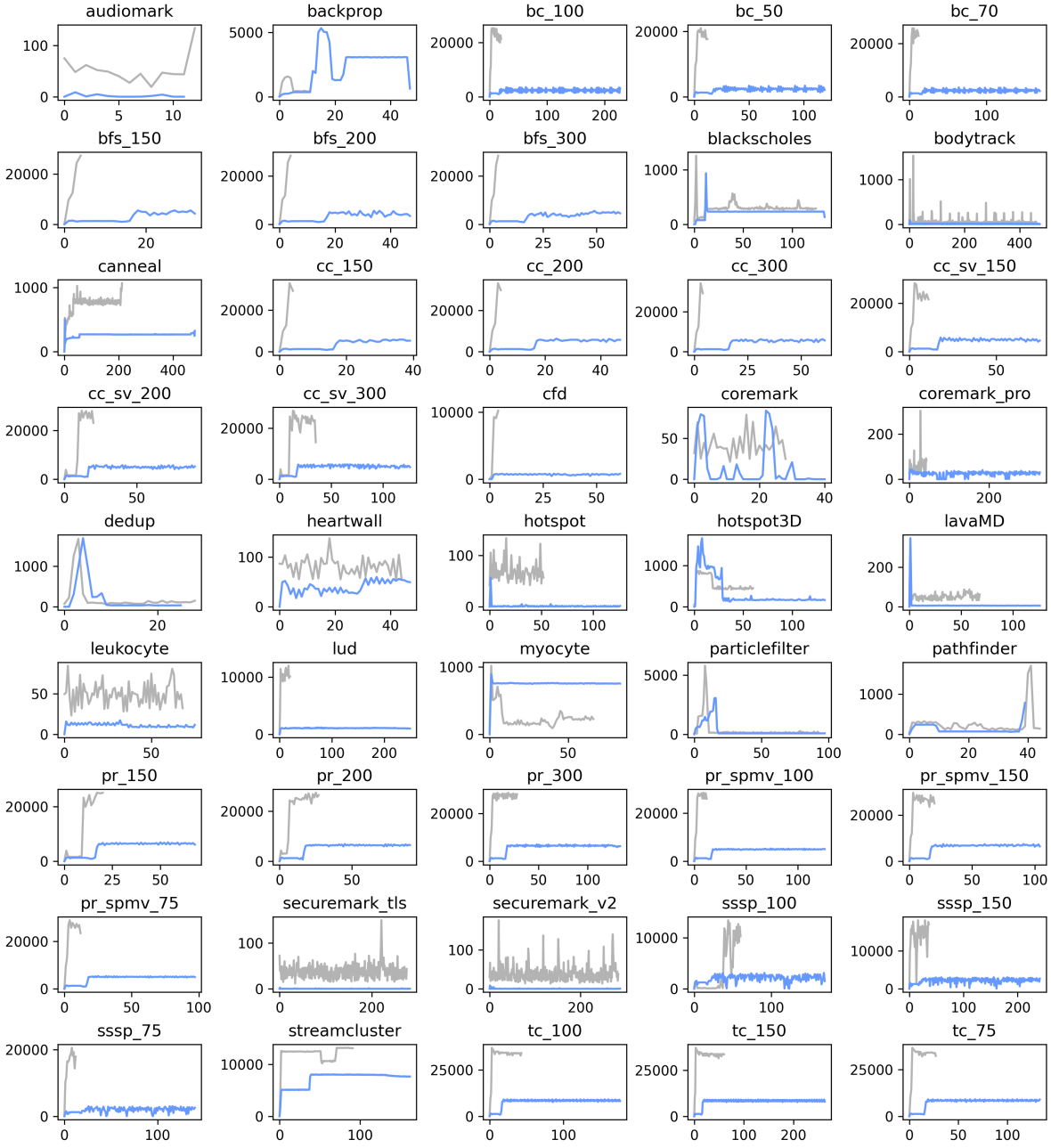


Figure 4.3.5: Memory read bandwidth (MB/s) of benchmarks on DRAM (gray) vs. Optane (blue).

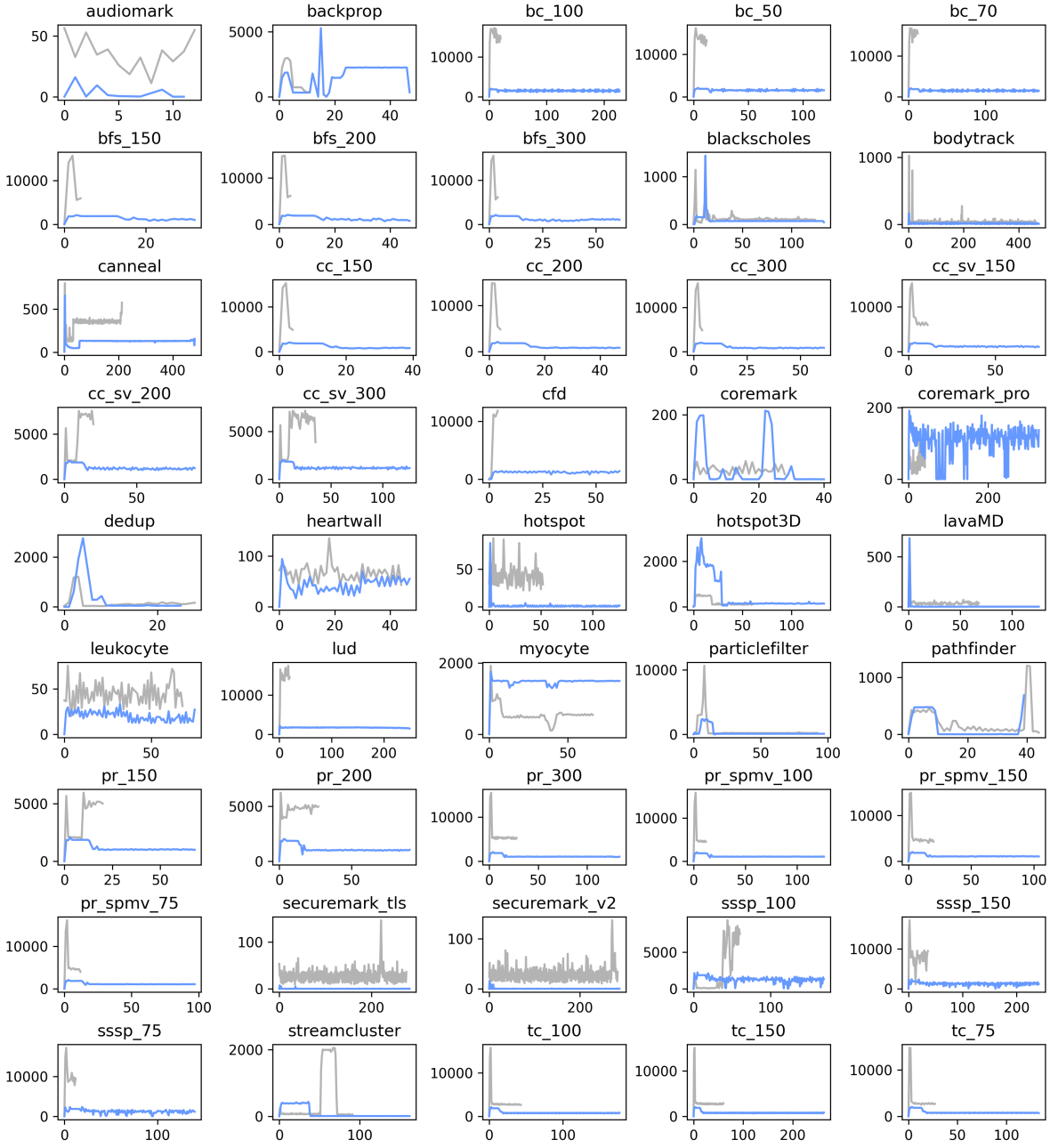


Figure 4.3.6: Memory write bandwidth (MB/s) of benchmarks on DRAM (gray) vs. Optane (blue).

4.3.5 Memory Accesses (Optane Only)

Since total memory accesses were only recorded for the Optane execution, Figure 4.3.7 presents the total memory read and write accesses observed.

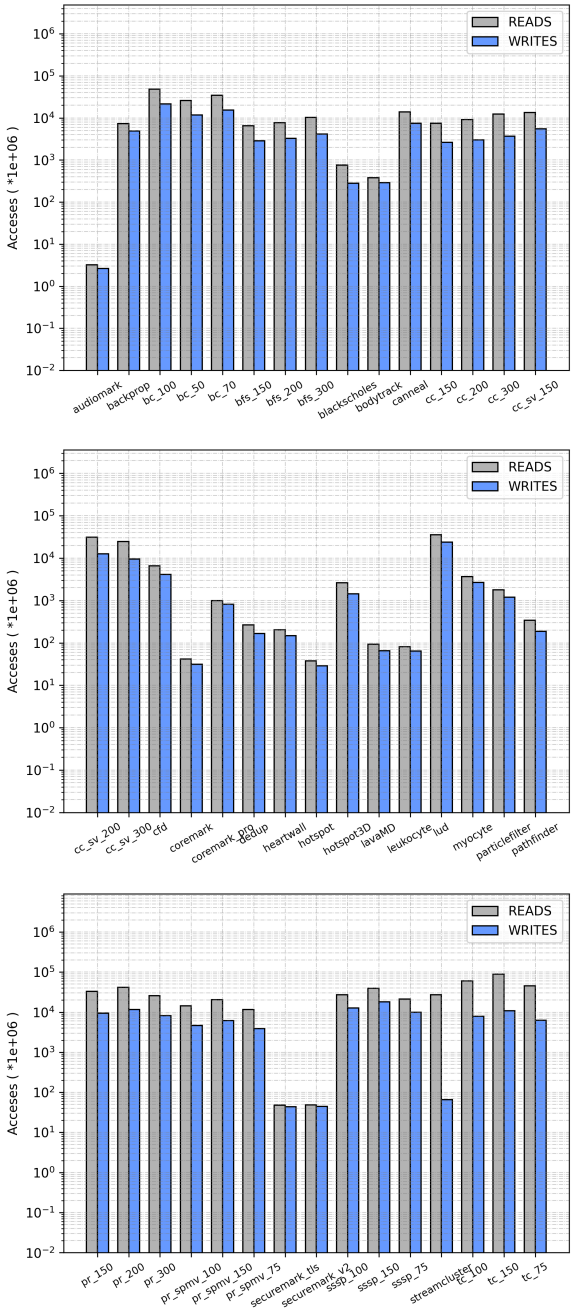


Figure 4.3.7: Total memory accesses (read and write) on Optane.

4.3.6 Observations

The profiling results show that Intel Optane generally has smaller memory bandwidth compared to DRAM, leading to slower execution times. However, it is observed that some benchmarks exhibit relatively close execution times on both memory systems, while others show significant differences,

with execution times being more than three times slower on Optane.

For the benchmarks with large execution time differences, it is found that they tend to have smaller IPC values and higher memory read and/or write bandwidths. These benchmarks are more sensitive to the reduced bandwidth of Optane, resulting in substantial slowdowns. In contrast, benchmarks with smaller execution time differences tend to have higher IPC values and lower memory bandwidth utilization, indicating they are less affected by the differences between DRAM and Optane.

These observations help differentiate between benchmarks that are heavily memory-bound and those that are more CPU-bound, providing essential insights for the classification process in the subsequent stages of the methodology.

4.4 Classification

To classify the benchmarks and prepare the dataset for machine learning models, specific thresholds are defined based on the profiling metrics, particularly memory bandwidth (Read BW and Write BW), and CPU utilization. These thresholds help to categorize the benchmarks into one of the four classes: Memory Intensive, Read Intensive, Write Intensive, and CPU Intensive.

Note: The metrics and thresholds mentioned in the classification process refer to the ones when the benchmarks were executed on Optane, as the performance characteristics on Optane were key to defining these thresholds.

4.4.1 Memory Intensive Class

The **Memory Intensive** class is defined by benchmarks where both memory read and write bandwidths (Read BW and Write BW) exceed a certain threshold. This threshold is set at the mean value of the Read BW and Write BW metrics, compared to half of the theoretical maximum. Specifically, a benchmark is categorized as Memory Intensive if both its Read BW and Write BW are above this threshold.

This criterion is used because benchmarks in this class are expected to heavily utilize the memory system, leading to significant memory throughput and high cache misses. By setting the threshold at half of the maximum Read and Write BW, we take into account that some benchmarks may utilize the memory bandwidth at full capacity only for a portion of their execution time. In these cases, during the remaining execution time, the memory bandwidth utilization may drop significantly or even be idle. However, these programs still exhibit significant memory demand during the periods of high utilization, which justifies their categorization as memory intensive.

4.4.2 Read Intensive Class

The **Read Intensive** class includes benchmarks that have a Read BW higher than the defined threshold, but a relatively low Write BW. These benchmarks focus on memory reads, meaning they require significant read throughput, but do not write large amounts of data to memory. Therefore, the defining characteristic for this class is that the Read BW must exceed the threshold, while the Write BW remains lower than that threshold.

This classification is useful for identifying benchmarks that are more focused on reading data from memory, which places specific demands on the memory hierarchy, especially in systems with large datasets or high memory read requirements.

4.4.3 Write Intensive Class

The **Write Intensive** class consists of benchmarks where the Write BW exceeds the threshold, while the Read BW remains relatively low. These benchmarks are characterized by a higher frequency of memory write operations, requiring significant write throughput. Like the Read Intensive class, the

Write Intensive class benchmarks place heavy demands on the memory system, but their focus is on writing data rather than reading it.

The threshold for this class ensures that only benchmarks with a substantial memory write load are categorized as Write Intensive, distinguishing them from those that are primarily focused on reads or computational tasks.

4.4.4 CPU Intensive Class

Finally, the **CPU Intensive** class includes benchmarks that do not meet the criteria for the previous three classes. These benchmarks are characterized by either low memory bandwidth utilization (both Read BW and Write BW are below the threshold) or high CPU usage, resulting in high instructions per cycle (IPC). CPU Intensive benchmarks focus more on computational tasks, and exhibit higher IPC than benchmarks in other classes, while their memory usage tends to be lower.

These benchmarks can either have low memory bandwidth utilization or have high IPC values, meaning their performance is primarily determined by the CPU rather than memory operations. This class serves as a catch-all category for benchmarks that do not heavily rely on memory but instead push the CPU to its limits.

4.4.5 Summary of Classification Criteria

The thresholds for classifying benchmarks into the four categories are summarized as follows:

- **Memory Intensive:** Benchmarks with both Read BW and Write BW above the threshold defined as the mean of these values compared to half of the maximum observed Read and Write BW.
- **Read Intensive:** Benchmarks with Read BW above the threshold but Write BW below it.
- **Write Intensive:** Benchmarks with Write BW above the threshold but Read BW below it.
- **CPU Intensive:** Benchmarks that do not meet any of the above criteria and primarily utilize the CPU.

These criteria enable the classification of each benchmark based on its memory and CPU usage patterns, creating a labeled dataset that can later be used for training machine learning models to predict the behavior of new benchmarks.

4.5 Machine Learning Models for Classification Prediction

Now that a labeled dataset has been created based on the profiling and classification of benchmarks into distinct categories (Memory Intensive, Read Intensive, Write Intensive, and CPU Intensive), the next step is to apply machine learning models to predict the class of new, unseen benchmarks. This process involves training supervised machine learning models using the labeled dataset, followed by evaluating their performance.

4.5.1 Supervised Learning Approach

In this research, supervised machine learning techniques are employed to classify benchmarks into their respective categories based on the features extracted during the profiling stage. The input features correspond to a known class label. The models are trained using this labeled dataset to learn the relationship between the features and class labels, allowing for predictions on new, unseen data.

A large number of features is included in the dataset to evaluate whether the models can correctly identify the benchmark classes regardless of the specific features used. In another scenario, classification might depend on different features beyond memory bandwidth, requiring a more flexible approach. Instead of relying solely on a predefined set of key metrics, the methodology proposed in this research

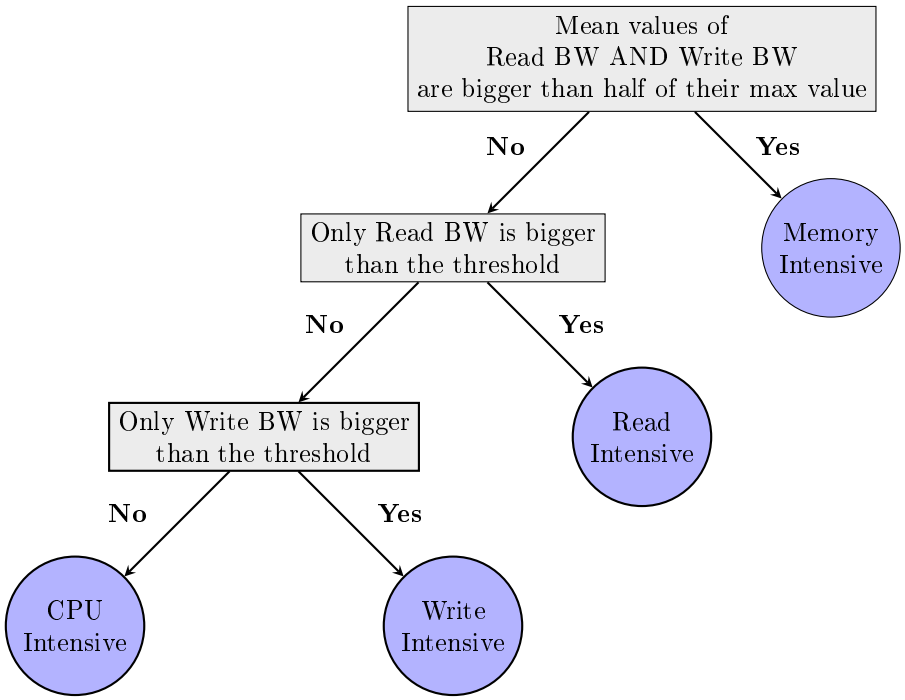


Figure 4.4.1: Diagram of the benchmark classification criteria.

is designed to work across different implementations, allowing for adaptability in various classification tasks.

Since many of the profiling metrics are collected as time-series data, the mean value of each metric is used as a representative feature. While full temporal graphs could provide more detailed insights, incorporating them directly into machine learning models would require complex architectures capable of processing sequential data. Using mean values simplifies implementation while still capturing the overall trends in benchmark behavior.

The features extracted for classification are:

- **Features extracted when running on DRAM:**
  - Execution time
  - Mean value of Instructions per Cycle (IPC)
  - Mean value of L3 cache hit ratio
  - Mean value of L3 cache misses per second
  - Mean value of read bandwidth
  - Mean value of write bandwidth
- **Features extracted when running on Optane:**
  - Execution time
  - Mean value of Instructions per Cycle (IPC)
  - Mean value of L3 cache hit ratio
  - Mean value of L3 cache misses per second
  - Mean value of read bandwidth

- Mean value of write bandwidth
- Total memory read accesses
- Total memory write accesses

Due to the high dimensionality of the feature set, dimensionality reduction techniques are applied to improve the performance of the models.

4.5.2 Dimensionality Reduction

Since machine learning models perform better with lower-dimensional data, dimensionality reduction is applied to reduce the feature space before training the models. This is particularly important because many of the original features are correlated, resulting in a reduction of the effectiveness of some models.

To achieve this, the Random Forest model is used to determine the feature importance in this process. Specifically, the most important features are selected such that their combined importance accounts for 70% of the total feature importance. This helps to retain the most relevant information while discarding less important features.

For further reduction, Principal Component Analysis (PCA) is applied to the selected important features. PCA projects the data onto a lower-dimensional space, retaining the maximum variance in the data. In this case, the dimensionality is reduced to two dimensions, which simplifies the problem and allows the models to perform better with fewer features.

4.5.3 Machine Learning Models

For this study, the following supervised learning models were selected:

- **Random Forests**: The number of trees used was 1000, with the splitting criterion set to 2, and the minimum node size set to 1.
- **K-Nearest Neighbors (KNN)**: The value of k was set to 4, using the Minkowski distance metric, and no weighting was applied.
- **Naive Bayes**

These models are chosen for their ability to handle classification tasks effectively, and they are well-suited supervised classification.

4.5.4 Training and Testing Procedure

The training and testing process follows these steps:

1. Split the dataset into training and testing sets and train a Random Forest model on the initial training data.
2. Identify the most important features that collectively contribute to 70% of the total feature importance using the trained Random Forest model.
3. Apply Principal Component Analysis (PCA) to further reduce the selected features to two dimensions.
4. Train the selected models (Random Forest, K-Nearest Neighbors and Naive Bayes) using the original feature set.
5. Evaluate the models on the test set using the specified evaluation metrics.
6. Split the dataset and train the selected models again using the reduced feature set obtained from feature importance selection.
7. Evaluate the models.

8. Split the dataset and train the selected models again using the reduced dataset produced by the PCA.
9. Evaluate the models.
10. Compare all models and feature reduction approaches, selecting the best-performing configuration based on the evaluation metrics.

To ensure reliable results, the entire training and testing process is repeated 10 times for each dataset split and training. This mitigates the impact of randomness in data selection. The final evaluation metrics are obtained by averaging the results across all iterations.

Each iteration uses a dataset split of 35 training samples and 10 testing samples.

4.6 Evaluation using a Class-Based Algorithm

In this section, we aim to evaluate the effectiveness of the class-based categorization methodology developed in the previous chapters. To achieve this, a multitasking environment is simulated, where multiple benchmarks are executed concurrently on the system.

A queue of benchmarks is generated, with each benchmark being introduced into the system according to a delay distribution. Different delay distributions, numbers of concurrent routines, and placement algorithms are tested to thoroughly assess the system's behavior.

Among the placement strategies evaluated, one is based on the class-oriented categorization produced by the machine learning model. This approach places tasks intelligently based on their predicted resource behavior. Additionally, several baseline placement algorithms are implemented for comparison, including random placement and round-robin placement.

This experimental setup provides insights into the benefits of intelligent workload placement in heterogeneous memory systems and highlights the advantages of class-based placement in dynamic multitasking scenarios.

4.6.1 Proposed Class-Based Placement Algorithm

The Class-Based Placement Algorithm was designed to efficiently manage tasks by considering the memory characteristics of each program. This algorithm leverages the classification of programs as determined by the model that has achieved the highest accuracy. Each task is classified into one of the following categories:

- **CPU-Intensive**
- **Memory-Intensive**
- **Read-Intensive**
- **Write-Intensive**

Once classified, tasks are assigned to either DRAM or PMEM based on their characteristics and the specific requirements of each memory type. The placement logic follows a set of rules designed to maximize system performance by balancing memory usage and exploiting the strengths of each memory type. Below is the reasoning behind the task assignment:

- **CPU-Intensive tasks** are always assigned to **PMEM**. CPU-Intensive tasks generally require minimal memory bandwidth, as their operations are primarily dependent on the processor's computation. PMEM (Optane) provides lower bandwidth compared to DRAM, but this is sufficient for tasks that do not demand high memory throughput. Assigning CPU-bound tasks to PMEM ensures that DRAM resources are preserved for more memory-demanding tasks. This allocation also avoids hindering the CPU-intensive tasks with slower memory access, as their performance is more dependent on computational power than memory speed.

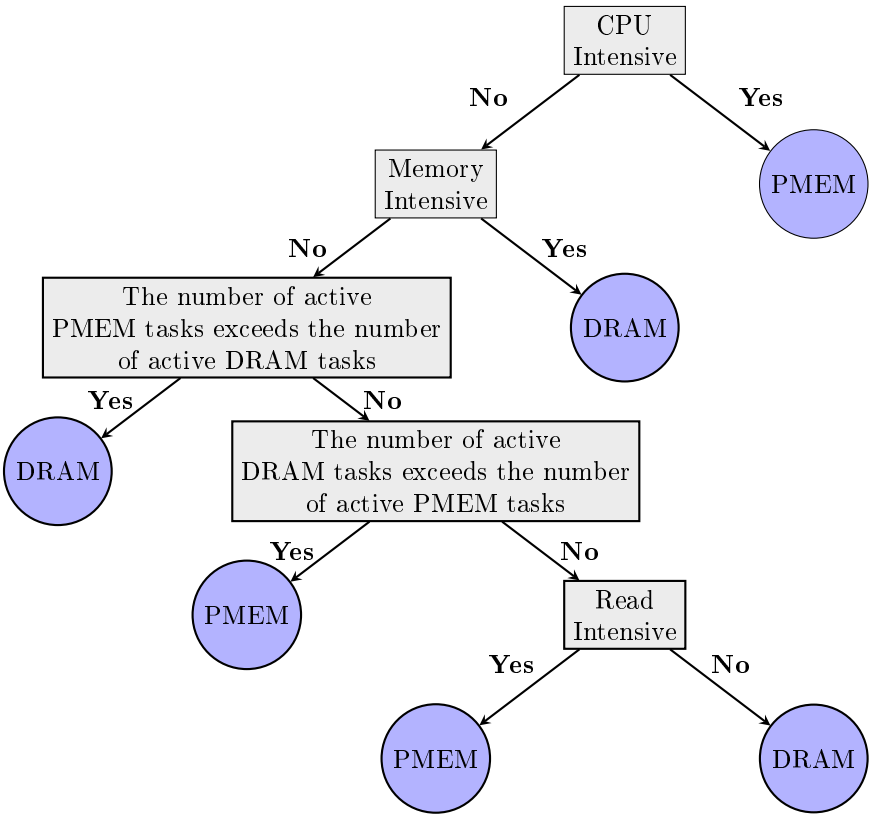


Figure 4.6.1: Diagram of Class-Based Placement Algorithm.

- **Memory-Intensive tasks** are always assigned to **DRAM**. Memory-intensive tasks require high memory throughput to operate efficiently. DRAM is significantly faster than PMEM, especially for tasks that involve frequent and large-scale memory reads and writes. These programs benefit from the high bandwidth and low latency provided by DRAM, which enables them to execute more rapidly. By assigning memory-intensive tasks to DRAM, we ensure that they are not bottlenecked by slow memory access and maximize their performance.
- **Read-Intensive tasks** are assigned based on the current system load: Read-intensive tasks involve frequent memory reads. While DRAM offers superior performance for read operations, PMEM handles reads more efficiently than writes and is therefore a suitable candidate when DRAM resources need to be preserved. To optimize memory utilization, read-intensive tasks are initially assigned to PMEM, reserving DRAM for tasks with more demanding memory requirements. However, if PMEM is currently experiencing a higher load, these tasks will instead be assigned to DRAM to maintain system balance and avoid overloading a single memory type.
- **Write-Intensive tasks** are assigned based on the current system load: Write-intensive tasks frequently update memory, requiring a memory type that can handle high write speeds. While DRAM is better suited for write-heavy workloads compared to PMEM, the algorithm considers the current memory load. If DRAM is heavily used, write-intensive tasks can be directed to PMEM, as long as the PMEM resources are not overloaded. In this way, the algorithm ensures that write-intensive tasks do not suffer from delays due to resource contention, balancing the system's performance based on available memory resources.

The algorithm follows a set of prioritized rules and executes the first rule that is satisfied:

- If the task is classified as **CPU-Intensive**, it is assigned to **PMEM**.
- If the task is classified as **Memory-Intensive**, it is assigned to **DRAM**.

- If the number of active tasks in **PMEM** exceeds those in **DRAM**, the task is assigned to **DRAM** to balance the load.
- If the number of active tasks in **DRAM** exceeds those in **PMEM**, the task is assigned to **PMEM**.
- If the number of active tasks in both **PMEM** and **DRAM** is equal, the decision is based on the task type:
  - **Read-Intensive tasks** are assigned to **PMEM**, as it handles read operations better than writes and preserves DRAM for more demanding tasks.
  - **Write-Intensive tasks** are assigned to **DRAM**, which offers superior write performance.

The goal of the algorithm is to optimize both **performance** and **resource utilization**. It ensures that each task is assigned to the most appropriate memory resource based on its specific characteristics. This helps prevent any memory resource from becoming a bottleneck, improving system efficiency and maximizing throughput by leveraging the strengths of both DRAM and PMEM.

4.6.2 Benchmark Arrival Distributions

To simulate a realistic multitasking environment, various statistical distributions are used to model the arrival times of benchmarks into the system. Specifically, three types of distributions are considered:

- **Uniform Distribution**
- **Gaussian (Normal) Distribution**
- **Poisson Distribution**

For all distributions, the mean arrival time is set to 1 minute, reflecting the average execution time observed across all benchmarks. A variance of 30 seconds is applied to introduce natural variability and simulate non-uniform arrival patterns.

In each experimental scenario, a batch of 10, 20, or 30 benchmarks is introduced into the system according to the respective delay distribution. This setup allows for testing under different workload intensities and timing conditions.

The use of diverse distributions ensures a comprehensive evaluation of the placement algorithms' robustness and adaptability under varying system loads and arrival patterns.

4.6.3 Placement Algorithms

In this study, five different placement algorithms are evaluated, each designed to allocate tasks to a single memory type - either DRAM or PMEM. The algorithms differ in how they assign tasks based on the memory demands of the programs. The following placement strategies are evaluated:

- **All in DRAM**: In this scenario, all programs are assigned to DRAM, regardless of their memory requirements. This simple placement approach does not take into account the different memory demands of the tasks, making it a baseline for comparison.
- **All in PMEM**: All programs are assigned to PMEM (Optane), independent of their memory demands. Like the "All in DRAM" approach, this method ignores the task classification and assigns all tasks to PMEM. This serves as another baseline to evaluate the performance when only one memory type is used for all tasks.
- **Random Assignment**: In this case, tasks are randomly assigned to either DRAM or PMEM. There is no consideration of the specific memory requirements or characteristics of the tasks. This method serves as a control to evaluate how well a random distribution of tasks performs compared to the other, more optimized methods.

- **Round Robin**: The Round Robin placement method alternates between assigning tasks to DRAM and PMEM in a cyclic manner. After each task is assigned, the placement assigns the next task to the opposite memory type. This method ensures that both DRAM and PMEM are used evenly across all tasks, providing a balanced approach to memory resource allocation.
- **Class-Based Algorithm**: The Class-Based Placement Algorithm classifies tasks based on their memory needs and assigns them to the appropriate memory type. CPU-Intensive tasks are assigned to PMEM, Memory-Intensive tasks to DRAM, and Read and Write Intensive tasks are assigned based on the current load and characteristics of the memory types. This algorithm aims to optimize system performance by using the strengths of each memory type for the corresponding tasks.

Each of these algorithms will be evaluated based on system performance, memory utilization, and task execution efficiency. The goal is to understand how different placement strategies affect overall system performance and resource allocation, and to compare these methods against one another to identify the most effective strategy for optimizing memory usage in a system that uses both DRAM and PMEM.

4.6.4 Performance Measurements

To evaluate the performance of the different placement algorithms, a set of key metrics are measured across the 15 runs (3 delay distributions x 5 placement algorithms). These measurements help to assess how well the system performs under different placement strategies, taking into account the total execution time, memory accesses, and memory utilization. The following metrics are used:

- **Total Execution Time**: The total execution time for all tasks under each placement algorithm. This is a primary metric for evaluating how efficiently the system processes the benchmarks. The goal is to minimize the execution time to ensure optimal performance.
- **PMEM Accesses**: The number of times PMEM (Optane) memory is accessed during the execution of the benchmarks. This metric is important to measure how often PMEM is being used and its efficiency in handling tasks. The goal is to minimize PMEM accesses to reduce latency and increase overall system performance.
- **Memory Utilization**: The total utilization of DRAM and PMEM memory during the execution of the tasks. This includes how much of each memory type is being actively used by the system. The objective is to maximize memory utilization, ensuring that both DRAM and PMEM are efficiently leveraged, without overloading either resource.
- **Task Execution Time Degradation**: The increase in the execution time of tasks when placed under non-ideal memory configurations. This metric helps to quantify how much slower a task becomes when not placed in the preferred memory (e.g., a memory-intensive task in PMEM). It provides insight into the placement algorithm's impact on individual task performance.

The performance evaluation aims to achieve the following goals:

- **Minimize Total Execution Time**: The primary goal is to reduce the total execution time, ensuring that tasks are completed as quickly as possible, regardless of the memory type assigned.
- **Maximize Memory Utilization**: The goal is to maximize the usage of both DRAM and PMEM, ensuring that resources are fully utilized without being under- or over-utilized. This allows the system to maintain high performance while efficiently balancing the workload across available memory types.
- **Minimize PMEM Accesses**: PMEM is slower than DRAM, so minimizing the number of accesses to PMEM is critical for maintaining performance. The fewer accesses to PMEM, the better the system can utilize its faster memory resources (DRAM), improving overall task execution efficiency. Additionally, frequent access to PMEM can reduce its lifespan over time. By minimizing PMEM accesses, we not only improve performance but also extend the operational lifespan of PMEM, ensuring that the system remains reliable and efficient in the long term.

- **Minimize Task Execution Time Degradation**: The aim is to reduce the extent to which tasks slow down due to suboptimal memory placement. By reducing degradation, the algorithm ensures fairer and more consistent performance across different task types. Minimizing execution time degradation also leads to increased task throughput, as more tasks can be completed in a given time frame, ultimately improving the overall efficiency of the system.

These measurements serve as the foundation for evaluating the effectiveness of the placement algorithms. By comparing these metrics across all experimental runs, we can determine which placement strategy provides the best balance between performance and resource usage.

Chapter 5

Experimental Evaluation

This chapter presents the experimental evaluation of the proposed methodology, aiming to assess the effectiveness of the classification process and to analyze the performance impact of different machine learning models and feature reduction techniques in a hybrid memory system context.

The evaluation begins with a detailed description of the hardware setup used to execute the benchmarks and collect profiling data. This includes the system configuration and the characteristics of the DRAM and PMEM memory modules.

Subsequently, the datasets used for classification are introduced. These datasets were created based on the profiling results and include labeled benchmark data categorized into performance-related classes. The categorization process and class definitions are explained in detail.

The next section evaluates the classification accuracy of the machine learning models. The impact of feature reduction is explored by comparing model performance across three scenarios: the original dataset, a reduced dataset using feature importance, and a PCA-transformed dataset. The best-performing model is selected based on accuracy for use in the final algorithm.

Finally, the classification is applied in a multitask scheduling environment using the proposed Class-Based algorithm. The scheduling performance is assessed through a series of experiments and compared against several baseline algorithms. The evaluation focuses on total execution time, PMEM accesses, memory utilization, and task execution degradation, offering insights into the effectiveness of class-based memory-aware placement.

5.1 Hardware Setup

The experiments were conducted on a system equipped with both DRAM and Intel Optane memory to evaluate benchmark performance under different memory configurations. The detailed hardware specifications are presented in Table 5.1.

Component	**Specification**
CPU	2x40 core Intel Xeon Gold 5218R CPU@2.10GHz
DRAM	4x32GB DDR4 DIMMs
Optane Memory	6x256GB Optane DC NVDIMMs

Table 5.1: Hardware Specifications of the Experimental System

The system was configured to run each benchmark under two memory setups: one using only DRAM and another using Intel Optane DIMMs. This allowed for a direct comparison of performance under

different memory architectures. The profiling process collected various execution metrics, which were later used for classification.

5.2 Dataset

In this section, we present the dataset with the features and their corresponding classifications. The benchmarks are categorized into four classes based on the profiling results. The table 5.2 displays the benchmarks, the 14 features extracted from profiling, and their corresponding classes.

The class assignments were made according to the criteria defined in Section 4.4.5. The dataset will serve as the foundation for the training and evaluation of the machine learning models in later sections of the methodology.

5.3 Classification Modeling

In this section, the results of the classification modeling are presented. First, the Random Forest model was used to identify the most important features. After reducing the dataset to the most important features, Principal Component Analysis (PCA) was applied to further reduce the dimensions to two.

5.3.1 Feature Reduction using Random Forest

The Random Forest model was trained on the original dataset, and the most important features were selected. The selected features were those that contributed to 70% of the total importance score. Notably, the most important features identified were the write bandwidth on DRAM, the L3 cache misses and the read and write bandwidth on Optane. This is significant because two out of these four features (read and write bandwidth on Optane) were used in the classification criteria to categorize the benchmarks into Memory-intensive, CPU-intensive, and other classes.

This demonstrates the effectiveness of the method, as the rejection of irrelevant information and retain of the most important features is achieved. The reduced dataset, with only the most important features, is shown in Table 5.3.

5.3.2 Dimensionality Reduction using PCA

Next, PCA was applied to the reduced dataset to further decrease the dimensions to two. This step allows for better visualization and facilitates model performance improvements. The resulting dataset with two dimensions is shown in Table 5.4.

The PCA coefficients for the two components are shown below in Table 5.5.

Benchmark	Exec. Time (s) (DRAM)	Exec. Time (s) (Optane)	Total Read Accesses	Total Write Accesses	Mean of IPC (DRAM)	Mean of L3 Hit Ratio (DRAM)	Mean of L3 Misses (KB) (DRAM)	Mean of IPC (Optane)	Mean of L3 Hit Ratio (Optane)	Mean of L3 Misses (KB) (Optane)	Mean of Read BW (MB/s) (DRAM)	Mean of Write BW (MB/s) (DRAM)	Mean of Read BW (MB/s) (Optane)	Mean of Write BW (MB/s) (Optane)	Class
audiomark	14	14	3221840	2625976	2.76	0.91	0.12	2.67	0.92	0.13	52.73	35.63	1.80	3.07	CPU-Intensive
backprop	19	115	7318057312	4870993168	0.90	0.51	31.96	0.23	0.38	4.23	731.99	1253.76	2295.69	1593.14	Write-Intensive
bc_100	24	238	48702891636	21574169128	0.10	0.67	229.15	0.03	0.64	20.03	20848.06	14796.15	2250.50	1521.48	Write-Intensive
bc_50	15	128	25794177240	11664292956	0.09	0.65	173.65	0.03	0.64	18.52	16463.00	12742.79	2166.79	1541.70	Write-Intensive
bc_70	17	168	34243579248	15322648844	0.11	0.66	213.34	0.03	0.65	19.49	19744.89	14361.35	2193.91	1511.68	Write-Intensive
bfs_150	5	37	6493820060	2867620796	0.11	0.31	40.19	0.06	0.50	6.20	14748.34	8239.68	2848.07	1369.46	Write-Intensive
bfs_200	7	46	7774855244	3292707468	0.11	0.30	45.60	0.07	0.47	6.74	15204.14	8319.72	3180.02	1221.94	Write-Intensive
bfs_300	9	66	10307747584	4149427444	0.11	0.32	47.17	0.07	0.45	7.35	15008.64	8313.88	3339.01	1156.06	Memory-Intensive
blackscholes	130	147	750413652	280392032	1.22	0.84	0.23	1.22	0.84	0.25	300.37	122.90	221.65	85.40	CPU-Intensive
bodytrack	486	487	376669332	289789060	2.30	0.89	0.15	2.31	0.89	0.14	60.54	36.30	7.78	12.10	CPU-Intensive
canneal	210	581	13920756940	7404236240	0.60	0.56	3.50	0.28	0.63	1.34	743.65	333.41	262.90	124.28	CPU-Intensive
cc_150	6	42	7514155720	2626160208	0.13	0.46	117.94	0.07	0.58	21.92	16968.44	7964.97	3513.98	1176.90	Memory-Intensive
cc_200	8	50	9148067080	2986749492	0.14	0.48	136.72	0.07	0.57	24.94	17323.85	7994.49	4006.93	1135.80	Read-Intensive
cc_300	11	68	12384236932	3672469532	0.14	0.50	156.32	0.06	0.56	27.91	17182.80	7974.98	4373.74	1078.68	Read-Intensive
cc_sv_150	15	73	13414068248	5461003144	0.22	0.88	85.86	0.07	0.90	13.75	19591.31	7342.54	4077.99	1273.66	Memory-Intensive
cc_sv_200	29	147	31014094024	12569779780	0.16	0.70	62.22	0.05	0.91	28.45	14787.66	4841.56	4281.51	1260.78	Memory-Intensive
cc_sv_300	37	134	24515150280	9569133068	0.19	0.79	62.02	0.07	0.94	15.01	16941.04	5162.22	4591.81	1236.25	Memory-Intensive
cfd	9	68	6532025036	4092119848	0.77	0.60	83.62	0.13	0.63	8.62	5830.37	6886.94	693.88	1190.81	Write-Intensive
coremark	35	42	41857880	31411452	2.30	0.89	0.18	2.24	0.89	0.19	41.91	29.73	13.47	33.51	CPU-Intensive
coremark_pro	44	332	993979320	819977076	1.83	0.91	0.17	1.70	0.86	0.44	61.20	48.97	24.80	107.14	CPU-Intensive
dedup	32	32	264994200	167961484	1.77	0.83	0.35	1.72	0.84	0.35	207.46	173.93	208.75	327.64	CPU-Intensive
heartwall	51	52	202196544	148545092	1.34	0.89	0.18	1.33	0.89	0.15	81.64	66.44	39.88	43.33	CPU-Intensive
hotspot3D	65	134	2611496620	1431452636	2.51	0.70	0.53	1.27	0.81	0.35	540.01	213.49	334.88	473.17	CPU-Intensive
hotspot	58	130	37831532	28700472	1.94	0.98	0.12	0.92	0.98	0.17	68.81	41.61	1.50	1.66	CPU-Intensive
lavaMD	69	130	92205248	64997412	2.23	0.90	0.14	1.22	0.89	0.14	55.56	43.95	8.65	6.59	CPU-Intensive
leukocyte	74	84	81562228	64387360	2.48	0.90	0.14	2.27	0.91	0.12	49.42	44.60	11.04	20.01	CPU-Intensive
lud	28	260	35872501456	24042900412	0.63	0.48	35.00	0.06	0.47	4.60	9599.63	13960.78	1066.44	1693.47	Write-Intensive
myocyte	75	83	3694425620	2652443732	1.84	0.80	0.35	1.59	0.86	0.19	240.11	549.50	747.70	1463.49	Write-Intensive
particlefilter	99	102	1792412144	1204383372	1.01	0.84	0.24	0.97	0.88	0.50	355.33	531.39	302.08	275.10	CPU-Intensive
pathfinder	42	43	344396152	187280604	2.10	0.82	0.36	1.98	0.85	0.26	255.57	205.94	128.01	124.92	CPU-Intensive
pr_150	26	147	33115837592	9401629924	0.11	0.68	66.59	0.04	0.90	39.32	12964.75	3706.36	5149.51	1176.2	Memory-Intensive
pr_200	29	186	41767421860	11688933076	0.13	0.79	78.24	0.04	0.91	41.72	19792.82	4612.38	5422.27	1134.29	Read-Intensive
pr_300	34	135	25745574560	8100801968	0.17	0.94	105.94	0.06	0.93	39.15	25353.96	5758.93	5825.22	1117.49	Read-Intensive
pr_spmv_100	18	76	14412748392	4628429540	0.17	0.91	79.69	0.07	0.91	24.26	22609.18	5859.65	4453.24	1156.69	Memory-Intensive
pr_spmv_150	27	105	20338593800	6182680012	0.17	0.93	78.69	0.06	0.93	26.38	24127.81	5375.00	5918.33	1165.56	Memory-Intensive
pr_spmv_75	14	60	11527878660	3889886648	0.16	0.89	73.16	0.07	0.90	22.95	22181.18	5824.57	4339.19	1175.46	Memory-Intensive
securemark_tls	280	283	48090296	43823496	3.08	0.90	0.14	3.08	0.91	0.11	38.24	28.00	0.16	0.15	CPU-Intensive
securemark_v2	286	287	48719028	44399188	2.90	0.91	0.13	2.90	0.93	0.09	38.93	29.93	0.21	0.19	CPU-Intensive
sssp_100	61	183	27287373296	12667658628	0.05	0.63	40.80	0.06	0.74	13.58	3713.87	2560.89	2062.70	1262.73	Write-Intensive
sssp_150	35	247	39339184984	17988201784	0.10	0.75	108.76	0.05	0.75	15.10	13655.53	7919.53	2173.16	1275.24	Write-Intensive
sssp_75	18	133	21377833440	9989568424	0.10	0.72	98.85	0.05	0.74	14.24	15321.63	9761.23	1929.89	1219.23	Write-Intensive
streamcluster	93	160	27045597248	65797840	0.65	0.41	10.94	0.40	0.52	3.12	12030.22	464.81	7222.04	94.46	Read-Intensive
tc_100	44	181	59872273456	7881400348	0.15	0.21	180.06	0.05	0.27	43.41	32312.04	3257.94	7822.89	838.12	Read-Intensive
tc_150	66	266	89368559832	10836359324	0.15	0.21	183.78	0.05	0.25	45.19	32530.12	3085.23	8013.28	808.99	Read-Intensive
tc_75	34	139	45571782064	6304694808	0.15	0.21	177.27	0.05	0.28	42.02	31336.56	3525.96	7581.57	866.71	Read-Intensive

Table 5.2: Dataset (Features and Labels)

BW (MB/s)	Mean of Write IPC (DRAM)	Mean of L3 Misses (KB) (Optane)	Mean of Read BW (MB/s) (Optane)	Mean of Write BW (MB/s) (Optane)	Class
audiomark	35.63	0.13	1.80	3.07	CPU-Intensive
backprop	1253.76	4.23	2295.69	1593.14	Write-Intensive
bc_100	14796.15	20.03	2250.50	1521.48	Write-Intensive
bc_50	12742.79	18.52	2166.79	1541.70	Write-Intensive
bc_70	14361.35	19.49	2193.91	1511.68	Write-Intensive
bfs_150	8239.68	6.20	2848.07	1369.46	Write-Intensive
bfs_200	8319.72	6.74	3180.02	1221.94	Write-Intensive
bfs_300	8313.88	7.35	3339.01	1156.06	Memory-Intensive
blackscholes	122.90	0.25	221.65	85.40	CPU-Intensive
bodytrack	36.30	0.14	7.78	12.10	CPU-Intensive
canneal	333.41	1.34	262.90	124.28	CPU-Intensive
cc_150	7964.97	21.92	3513.98	1176.90	Memory-Intensive
cc_200	7994.49	24.94	4006.93	1135.80	Read-Intensive
cc_300	7974.98	27.91	4373.74	1078.68	Read-Intensive
cc_sv_150	7342.54	13.75	4077.99	1273.66	Memory-Intensive
cc_sv_200	4841.56	28.45	4281.51	1260.78	Memory-Intensive
cc_sv_300	5162.22	15.01	4591.81	1236.25	Memory-Intensive
cfd	6886.94	8.62	693.88	1190.81	Write-Intensive
coremark	29.73	0.19	13.47	33.51	CPU-Intensive
coremark_pro	48.97	0.44	24.80	107.14	CPU-Intensive
dedup	173.93	0.35	208.75	327.64	CPU-Intensive
heartwall	66.44	0.15	39.88	43.33	CPU-Intensive
hotspot3D	213.49	0.35	334.88	473.17	CPU-Intensive
hotspot	41.61	0.17	1.50	1.66	CPU-Intensive
lavaMD	43.95	0.14	8.65	6.59	CPU-Intensive
leukocyte	44.60	0.12	11.04	20.01	CPU-Intensive
lud	13960.78	4.60	1066.44	1693.47	Write-Intensive
myocyte	549.50	0.19	747.70	1463.49	Write-Intensive
particlefilter	531.39	0.50	302.08	275.10	CPU-Intensive
pathfinder	205.94	0.26	128.01	124.92	CPU-Intensive
pr_150	3706.36	39.32	5149.51	1176.22	Memory-Intensive
pr_200	4612.38	41.72	5422.27	1134.29	Read-Intensive
pr_300	5758.93	39.15	5825.22	1117.49	Read-Intensive
pr_spmv_100	5859.65	24.26	4453.24	1156.69	Memory-Intensive
pr_spmv_150	5375.00	26.38	5918.33	1165.56	Memory-Intensive
pr_spmv_75	5824.57	22.95	4339.19	1175.46	Memory-Intensive
securemark_tls	28.00	0.11	0.16	0.15	CPU-Intensive
securemark_v2	29.93	0.09	0.21	0.19	CPU-Intensive
sssp_100	2560.89	13.58	2062.70	1262.73	Write-Intensive
sssp_150	7919.53	15.10	2173.16	1275.24	Write-Intensive
sssp_75	9761.23	14.24	1929.89	1219.23	Write-Intensive
streamcluster	464.81	3.12	7222.04	94.46	Read-Intensive
tc_100	3257.94	43.41	7822.89	838.12	Read-Intensive
tc_150	3085.23	45.19	8013.28	808.99	Read-Intensive
tc_75	3525.96	42.02	7581.57	866.71	Read-Intensive

Table 5.3: Reduced Dataset after Random Forest Feature Selection

Benchmark	PCA Component 1	PCA Component 2	Class
audiomark	-2.17	-0.21	CPU-Intensive
backprop	-0.01	0.49	Write-Intensive
bc_100	1.92	1.8	Write-Intensive
bc_50	1.65	1.6	Write-Intensive
bc_70	1.84	1.77	Write-Intensive
bfs_150	0.71	1.1	Write-Intensive
bfs_200	0.68	0.91	Write-Intensive
bfs_300	0.67	0.81	Memory-Intensive
blackscholes	-2.04	-0.19	CPU-Intensive
bodytrack	-2.16	-0.21	CPU-Intensive
canneal	-1.93	-0.17	CPU-Intensive
cc_150	1.22	0.3	Memory-Intensive
cc_200	1.39	0.07	Read-Intensive
cc_300	1.52	-0.14	Read-Intensive
cc_sv_150	1.06	0.41	Memory-Intensive
cc_sv_200	1.35	-0.43	Memory-Intensive
cc_sv_300	0.94	-0.06	Memory-Intensive
cfd	0.07	1.17	Write-Intensive
coremark	-2.14	-0.19	CPU-Intensive
coremark_pro	-2.06	-0.15	CPU-Intensive
dedup	-1.81	-0.01	CPU-Intensive
heartwall	-2.12	-0.19	CPU-Intensive
hotspot3D	-1.65	0.08	CPU-Intensive
hotspot	-2.17	-0.21	CPU-Intensive
lavaMD	-2.16	-0.21	CPU-Intensive
leukocyte	-2.15	-0.2	CPU-Intensive
lud	1.19	2.54	Write-Intensive
myocyte	-0.65	0.75	Write-Intensive
particlefilter	-1.8	-0.02	CPU-Intensive
pathfinder	-2.01	-0.13	CPU-Intensive
pr_150	1.73	-1.17	Memory-Intensive
pr_200	1.92	-1.2	Read-Intensive
pr_300	2.02	-1.07	Read-Intensive
pr_spmv_100	1.25	-0.28	Memory-Intensive
pr_spmv_150	1.58	-0.72	Memory-Intensive
pr_spmv_75	1.19	-0.2	Memory-Intensive
securemark_tls	-2.17	-0.21	CPU-Intensive
securemark_v2	-2.17	-0.21	CPU-Intensive
sssp_100	0.13	0.19	Write-Intensive
sssp_150	0.78	0.86	Write-Intensive
sssp_75	0.85	1.15	Write-Intensive
streamcluster	-0.48	-1.73	Read-Intensive
tc_100	2.06	-2.17	Read-Intensive
tc_150	2.12	-2.31	Read-Intensive
tc_75	2.02	-2.02	Read-Intensive

Table 5.4: Dataset after PCA Dimensionality Reduction

Feature	PCA Component 1	PCA Component 2
Mean value of L3 cache misses / second (Optane)	0.521886078	-0.440656298
Mean value of Write BW (DRAM)	0.461346622	0.592826902
Mean value of Read BW (Optane)	0.495225754	-0.527654660
Mean value of Write BW (Optane)	0.519177876	0.419474491

Table 5.5: PCA Coefficients

5.3.3 PCA Visualization and SVM Classification

After performing Principal Component Analysis (PCA), the dataset was reduced to two dimensions. This allowed for a clear visual representation of the benchmark classes in the reduced space. The following figure shows the PCA visualization, with each point representing a benchmark. The color of the points indicates their class, and it is evident that the benchmarks are well distributed in distinct regions within the two-dimensional space.

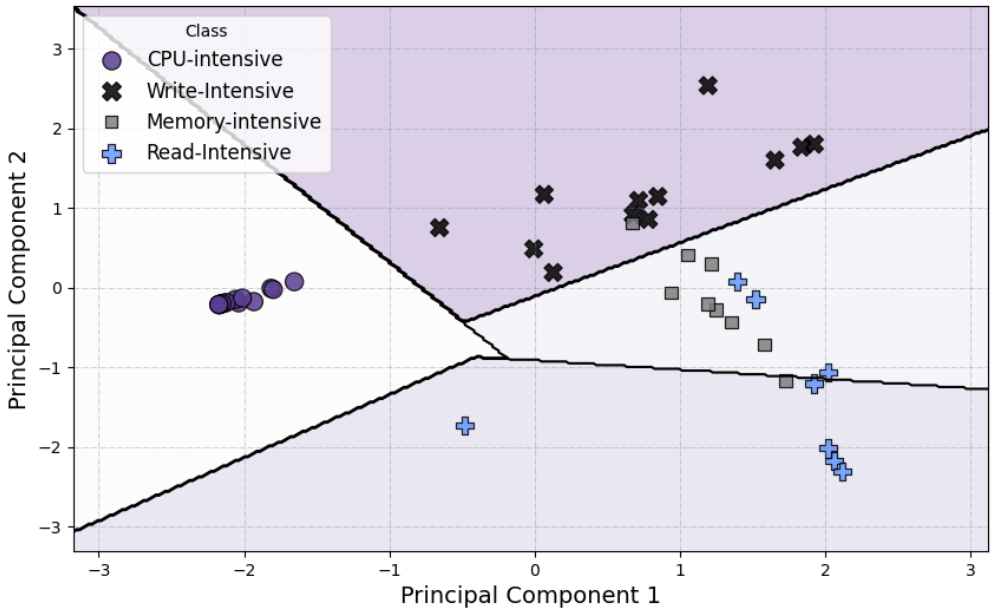


Figure 5.3.1: PCA Visualization of the Dataset and SVM Classification

From the plot, it is observed that the different classes are grouped together in separate areas. This distribution suggests that the benchmarks of the same class share similar characteristics, and these patterns are clearly visible in the PCA plot. The SVM model, when applied to this reduced two-dimensional space, can easily draw linear boundaries between the classes. The well-defined regions make it easier for the classifier to distinguish between the classes based on the extracted features.

These observations demonstrate that PCA has successfully captured the underlying structure of the data in just two dimensions, and the SVM model performs well in separating the benchmarks into their respective classes. This suggests that other models, such as Random Forest, KNN and Naive Bayes, will likely perform well with this dimensionality reduction, as the data is now in a space that clearly distinguishes between the different classes.

5.3.4 Training and Evaluation with the Original Dataset

The first step involved training the models using the original dataset. The performance was evaluated using only the accuracy metric. The accuracy results for each model are presented below.

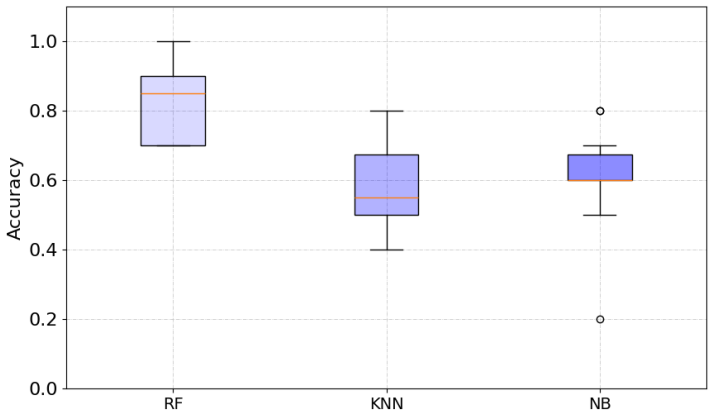


Figure 5.3.2: Box plots of accuracy on the Original Dataset

5.3.5 Model Training with the Reduced Features Dataset

After using Random Forest to reduce the dataset, the models were retrained on the reduced feature set. The evaluation metrics for this dataset are shown in the same format below.

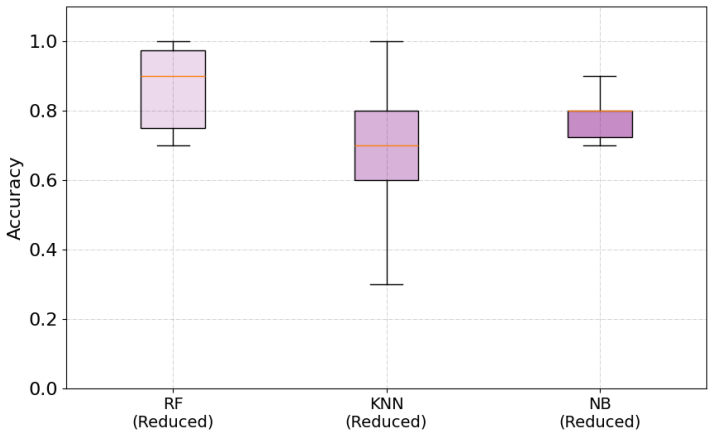


Figure 5.3.3: Box plots of accuracy on the Reduced Dataset

5.3.6 Model Training with the PCA-Reduced Dataset

Next, PCA was applied to reduce the dimensionality of the dataset to two, and the models were retrained on the PCA-reduced dataset. The evaluation metrics for this dataset are also shown below.

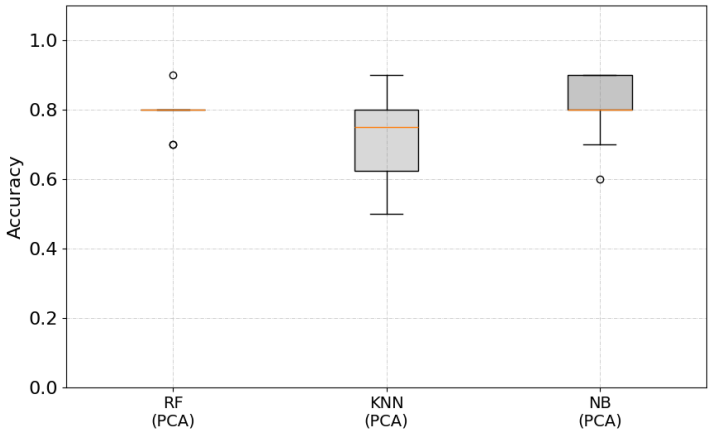


Figure 5.3.4: Box plots of accuracy on the PCA-Reduced Dataset

5.3.7 Summarized Results

The models were evaluated using three different scenarios: the original dataset, the important features dataset, and the PCA-reduced dataset.

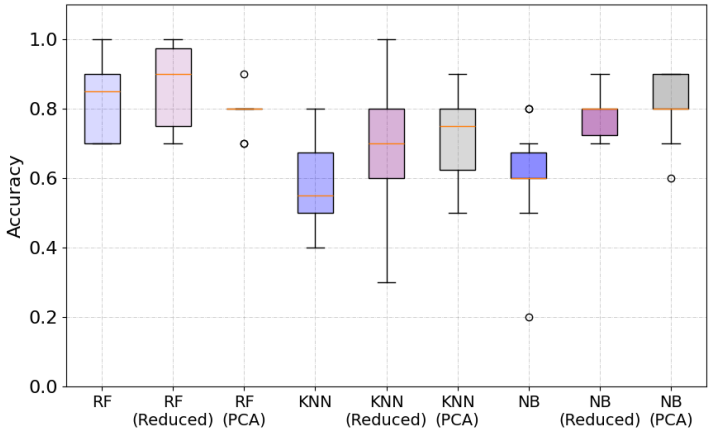


Figure 5.3.5: Box plots of accuracy of the models and datasets

The mean accuracy for each of these scenarios was calculated and the results are summarized in the table below.

Scenario	Random Forest	KNN	Naive Bayes
Original Dataset	82%	58%	60%
Important Features Dataset	87%	68%	79%
PCA-Reduced Dataset	79%	72%	81%

Table 5.6: Mean accuracy values for all the evaluation scenarios.

The results in Table 5.6 reveal several noteworthy trends. First, the Random Forest model consistently

performed well, achieving 82% accuracy on the original dataset, which increased to 87% with important feature selection and remained high at 79% after PCA reduction. In contrast, the KNN model initially struggled on the original dataset with an accuracy of 58%, but its performance improved significantly to 68% when only the important features were used, and further to 72% after PCA reduction. Similarly, Naive Bayes marked an improvement from 60% on the original dataset to 79% with important feature selection, and then to 81% with PCA reduction.

These observations indicate that reducing the feature space - first by selecting the most relevant features and then by applying PCA - not only helps to eliminate irrelevant information but also to substantially improve the performance of models that are more sensitive to high-dimensional data. In particular, the marked improvement for KNN and Naive Bayes suggests that this reduction method effectively captures the underlying structure of the data.

Based on the accuracy values presented, the best performance was achieved by the Random Forest model on the Important Features Dataset, with an accuracy of 87%. Consequently, the Random Forest model trained on the Important Features Dataset will be used for further evaluation with the multitask algorithm.

5.4 Evaluation of Results using a Class-Based Algorithm

This section presents the results of the experimental evaluation, focusing on the performance of the proposed Class-Based Placement Algorithm in comparison with alternative placement strategies. The goal is to assess how effectively each algorithm manages task scheduling and memory assignment in a hybrid DRAM-PMEM environment.

The evaluation considers key performance metrics for each placement strategy. These metrics provide comprehensive insights into the overall system efficiency, resource usage, and the impact of scheduling decisions on task performance.

Experiments were conducted using three different delay distributions - **Uniform**, **Gaussian**, and **Poisson** - to simulate varying task arrival patterns. Each of the five placement algorithms was tested under these distributions, resulting in a total of 15 experimental runs. For each algorithm, the results across the different distributions were averaged to produce a representative performance evaluation. The resulting measurements are illustrated in the following figures.

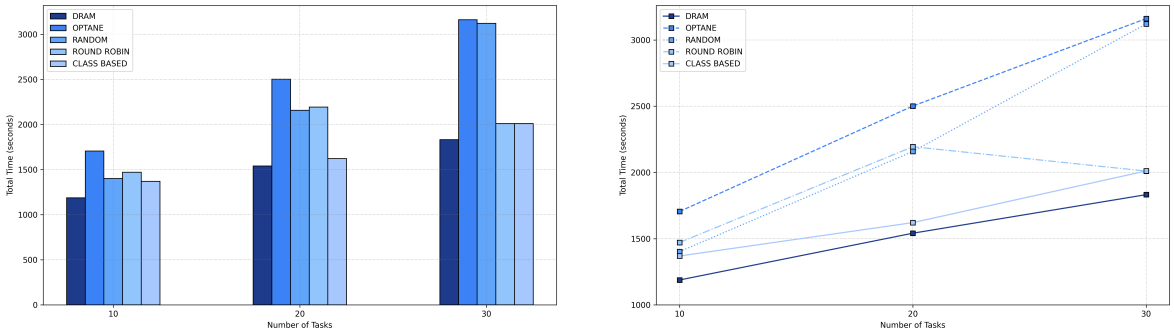
5.4.1 Total Execution Time

The Total Execution Time is a critical metric for evaluating the efficiency of the placement algorithms. It measures how quickly the system processes all incoming tasks, with lower values indicating better overall performance. In this experiment, we compare the execution times across all placement algorithms by averaging the results obtained under the different delay distributions. The findings are illustrated in Figures 5.4.1a and 5.4.1b.

From the bar plots, several observations can be made. As expected, the **DRAM-only** algorithm delivers the best execution times across all task volumes, while the **PMEM-only** approach results in the highest execution times due to PMEM's slower performance.

The **Random Assignment** and **Round Robin** algorithms produce intermediate results, balancing load but not optimizing memory performance characteristics. The proposed **Class-Based Placement Algorithm** demonstrates a substantial improvement in execution time over these generic strategies. Although not outperforming the DRAM-only case, it significantly reduces total execution time compared to Random, Round Robin, and PMEM-only placement.

Additionally, the **Class-Based Algorithm** scales more gracefully with increasing task counts - the growth rate of execution time is lower compared to Random, Round Robin, and PMEM-only placement. Notably, the Round Robin algorithm shows a surprising dip in execution time when running 30



(a) Total Execution Time for different placement algorithms.

(b) Total Execution Time vs Number of Tasks.

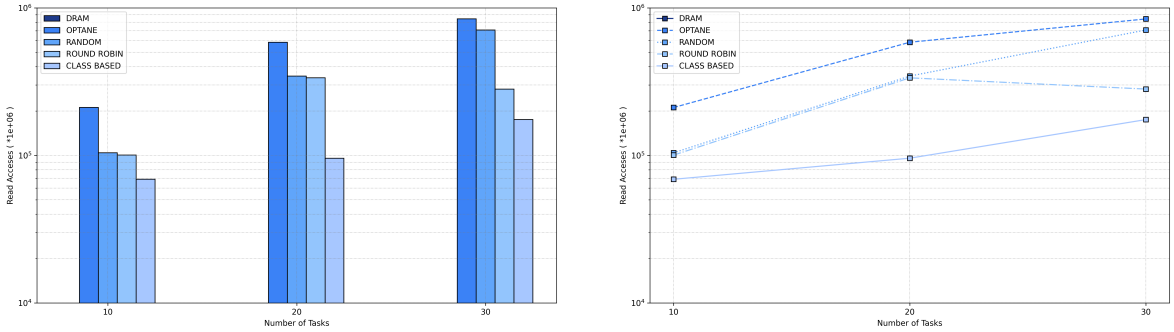
Figure 5.4.1: Comparison of total execution time across placement algorithms. Left: aggregated total times. Right: scaling behavior with increasing number of tasks.

tasks, reaching performance close to that of the Class-Based Algorithm. This anomaly will be further discussed after the presentation of all evaluation metrics.

5.4.2 Read and Write Accesses

Another important performance metric is the number of memory accesses - specifically, how often PMEM is read from or written to. These accesses significantly impact both execution time and memory longevity, particularly due to PMEM's slower speed and lower write endurance compared to DRAM.

Figures 5.4.2 and 5.4.3 present the Read and Write Access patterns, respectively. Each figure includes two plots: a bar chart comparing total accesses across placement algorithms, and a line plot illustrating how accesses scale with the number of tasks.



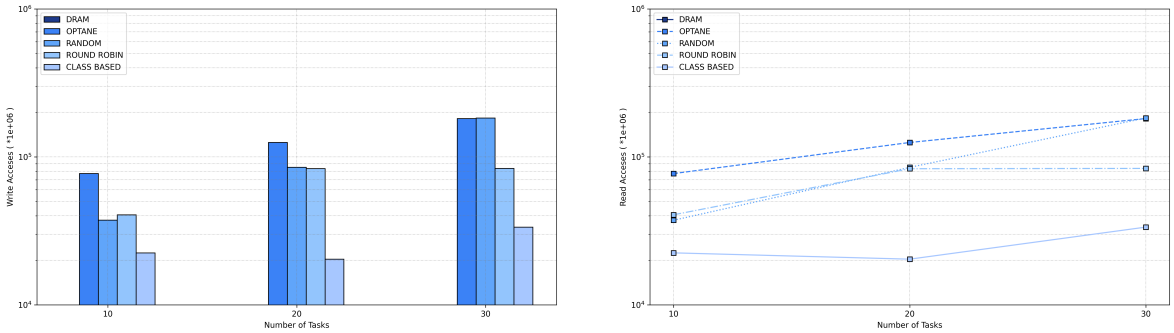
(a) Total Read Accesses across placement algorithms.

(b) Read Accesses vs Number of Tasks.

Figure 5.4.2: Comparison of Read Access patterns. Left: aggregated Read Accesses. Right: how accesses scale with increasing number of tasks.

The results show that the Class-Based Placement Algorithm significantly reduces PMEM read pressure compared to Optane-only, Random, and Round Robin strategies. DRAM-only placement shows zero PMEM accesses, as expected. Both Random and Round Robin placement exhibit intermediate behavior between the Optane-only and Class-Based approaches.

For write accesses, a similar trend is observed. DRAM-only placement results in zero PMEM writes, while Optane-only exhibits the highest write volume. Again, Random and Round Robin strategies fall



(a) Total Write Accesses across placement algorithms.

(b) Write Accesses vs Number of Tasks.

Figure 5.4.3: Comparison of Write Access patterns. Left: aggregated Write Accesses. Right: how accesses scale with increasing number of tasks.

in between, with the Class-Based Algorithm achieving a significant reduction in PMEM write traffic. This not only enhances system responsiveness but also helps prolong the PMEM lifespan.

As with read accesses, the Round Robin placement approach shows an unusual stagnation between 20 and 30 tasks, indicating that this behavior will be explained along with the time constancy analysis.

5.4.3 Memory Utilization

Memory Utilization offers a deeper understanding of how effectively the system leverages available memory bandwidth in both DRAM and PMEM throughout execution. This metric is split into four components: DRAM Read Bandwidth, DRAM Write Bandwidth, PMEM Read Bandwidth, and PMEM Write Bandwidth.

For each of the selected placement algorithms - Random, Round Robin, and Class-Based - we assess utilization at three different task volumes: 10, 20, and 30 tasks. The results are illustrated using pie charts, where each pie represents the average bandwidth usage over time, normalized to a baseline. The baseline is defined as 100% utilization, calculated from the average bandwidth observed when all tasks were forced to execute entirely in DRAM or entirely in PMEM, respectively.

Each pie chart, therefore, provides a relative measure of how much of the maximum potential bandwidth (in either DRAM or PMEM) was utilized by a particular algorithm during the experiment. This method enables a fair comparison across memory types and placement policies.

At 10 tasks, we observe that all three placement algorithms exhibit similar memory utilization, hovering around 50%. This indicates a fairly balanced use of both DRAM and PMEM across the different strategies. This balanced utilization is expected at lower task counts, where the system's memory demand is not yet overwhelming.

As the task count increases, the utilization patterns begin to diverge. The Random placement algorithm shows a rise in PMEM utilization due to the random assignment of memory-intensive tasks. This increased reliance on PMEM introduces limitations as the task load grows, which helps explain the performance challenges associated with this algorithm.

The Round Robin algorithm, in contrast, slightly reduces its DRAM utilization while increasing its PMEM usage. By 30 tasks, the system shows more stable access patterns, which reflects an efficient balancing of memory resources. This gradual adjustment suggests that Round Robin can manage the growing number of tasks by making better use of PMEM, and this is aligned with the decrease in execution time and task access patterns observed earlier.

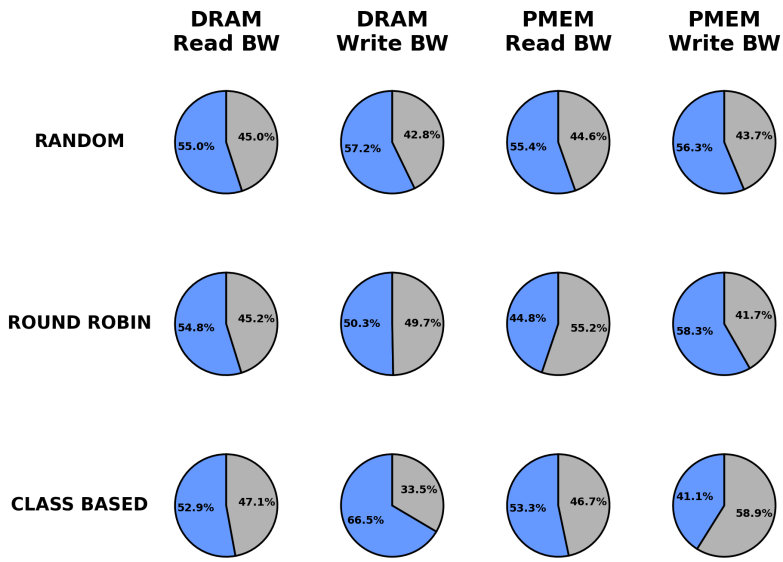


Figure 5.4.4: Memory Utilization (10 Tasks) across DRAM and PMEM for Random, Round Robin, and Class-Based Placement Algorithms.

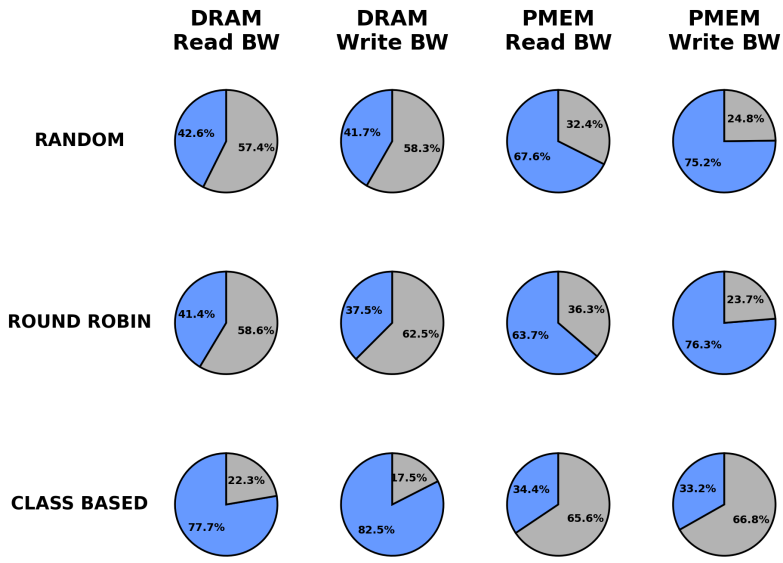


Figure 5.4.5: Memory Utilization (20 Tasks) across DRAM and PMEM for Random, Round Robin, and Class-Based Placement Algorithms.

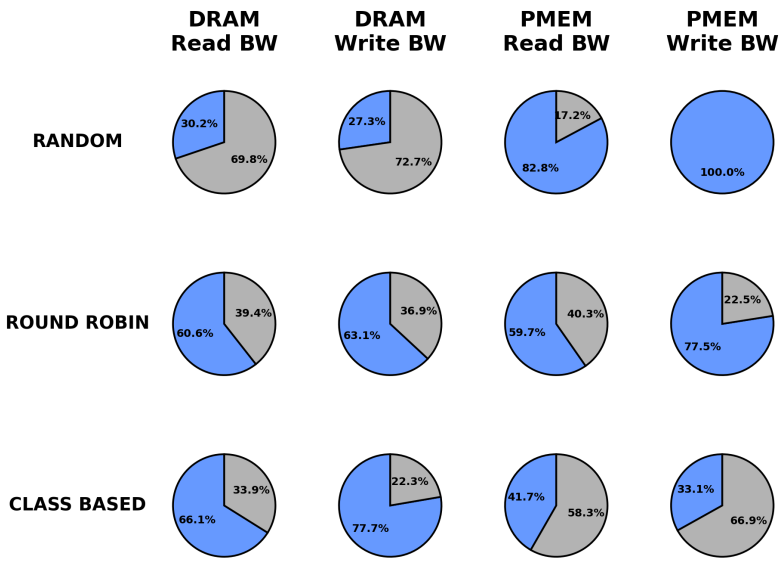


Figure 5.4.6: Memory Utilization (30 Tasks) across DRAM and PMEM for Random, Round Robin, and Class-Based Placement Algorithms.

The Class-Based placement algorithm presents the most notable shift. While DRAM utilization increases slightly, PMEM usage decreases as the task load grows. This shift indicates that Class-Based favors DRAM over PMEM as the workload increases, optimizing memory usage for better system performance. As the load increases, this strategy adapts by relying on DRAM for more efficient task management, ensuring that the system operates smoothly and avoids bandwidth saturation in PMEM. This behavior shows that the Class-Based algorithm is better suited for handling large task loads compared to the other algorithms, offering a more balanced and optimized memory resource allocation.

5.4.4 Task Execution Time Degradation

Task Execution Time Degradation provides a detailed view of how each placement algorithm impacts the performance of individual tasks relative to their ideal execution time on DRAM. To evaluate this, we use box plots showing the normalized execution time for each task, where normalization is done against the task's execution time when run entirely in DRAM.

The box plots in Figure 5.4.7 present the execution time degradation of tasks under three different placement algorithms - Random, Round Robin, and Class-Based. The execution times are normalized to the DRAM-only baseline to highlight the deviation each task experiences when not executed in DRAM.

As the number of tasks increases from 10 to 30, the Random placement algorithm exhibits a noticeable increase in variance. This indicates that more tasks suffer from higher degradation, likely due to arbitrary assignment to PMEM even when it negatively impacts performance.

The Round Robin algorithm follows a similar trend, with variance growing from 10 to 20 tasks. However, at 30 tasks, the variance decreases, aligning with earlier observations in the execution time and memory access results. This suggests that at higher loads, Round Robin unintentionally achieves a more balanced use of PMEM and DRAM.

The Class-Based Placement Algorithm consistently shows the lowest execution time variance across all task counts. The increase in variance as tasks row is minimal, indicating that the algorithm successfully

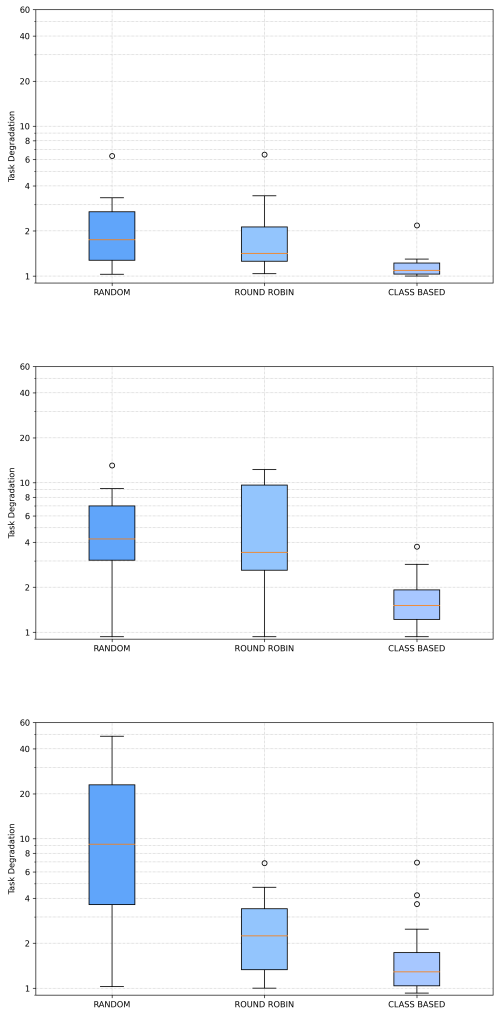


Figure 5.4.7: Task Execution Time Degradation. Execution times are normalized to DRAM-only execution. The top plot corresponds to 10 tasks, the middle to 20 tasks, and the bottom to 30 tasks.

mitigates performance degradation. This stable behavior reflects the model's ability to place tasks in memory types more suited to their profiles, preserving throughput and minimizing execution delays.

We can now interpret the observed behavior of the Round Robin algorithm at 30 tasks. While the memory utilization results show increased use of PMEM, the Task Degradation plots reveal that Round Robin maintains a relatively low task throughput. This combination suggests that, in the middle of the execution, a significant portion of tasks were written to PMEM. However, by chance, many of these tasks were not highly memory-intensive, resulting in fewer overall accesses despite heavy PMEM usage.

The seemingly improved total execution time at 30 tasks can therefore be attributed to the fact that execution time is heavily influenced by the duration of the last few tasks. It appears that, coincidentally, the final tasks were assigned in a manner similar to the Class-Based algorithm, leading to better performance in that specific case.

In conclusion, the improved result of Round Robin at 30 tasks is not a reflection of consistently efficient placement, but rather a fortunate outcome caused by the specific task assignments at the end of the workload. Unlike the Class-Based algorithm, Round Robin does not provide effective load balancing

throughout the entire execution.

5.4.5 Summary of the Evaluation Results

The experimental evaluation demonstrates the effectiveness of the proposed **Class-Based Placement Algorithm**, which leverages classification to make informed placement decisions based on task characteristics and system state.

The Class-Based Placement Algorithm was compared against four baseline algorithms: **DRAM-only**, **PMEM-only**, **Random assignment**, and **Round Robin**.

Across multiple metrics and experimental scenarios, the Class-Based approach demonstrated robust advantages:

- **Execution Time**: The Class-Based algorithm consistently achieved lower total execution times compared to Random, Round Robin, and PMEM-only strategies, second only to the DRAM-only setup which is idealized and not scalable.
- **Memory Accesses**: It significantly reduced PMEM accesses by favoring DRAM when appropriate, thereby preserving PMEM lifespan and reducing latency.
- **Memory Utilization**: The Class-Based algorithm balanced memory usage effectively, adapting to load and task types. As the number of tasks increased, it leaned more on DRAM, showing intelligent resource management.
- **Task Execution Time Degradation**: It consistently showed the smallest variance in task degradation, maintaining uniform task performance and high throughput. This implies better scheduling consistency and more predictable execution times.

These results confirm that combining classification with resource-aware placement leads to improved performance, better memory utilization, and more efficient use of hybrid memory systems. The Class-Based Placement Algorithm not only optimizes for performance but also contributes to system longevity and operational efficiency, making it a strong candidate for real-world deployment in heterogeneous memory environments.

Chapter 6

Conclusion and Future Work

This research presented a comprehensive methodology for classifying benchmarks based on their memory and CPU utilization profiles, with the ultimate goal of leveraging machine learning models for efficient categorization. The approach involved the data collection through profiling, feature extraction, training machine learning models, and evaluation using various performance metrics. We focused on four benchmark classes - Memory Intensive, Read Intensive, Write Intensive, and CPU Intensive - based on the memory and CPU bandwidth usage characteristics during execution.

The methodology utilized supervised learning models, including Random Forest, K-Nearest Neighbors (KNN), and Naive Bayes, which were trained on the original benchmark data, as well as reduced feature sets. Through feature importance analysis, we reduced the number of features, retaining only those deemed most important for classification. Principal Component Analysis (PCA) was also employed to further reduce the dimensionality of the feature space, allowing for a two-dimensional visualization and an improved classification performance.

Our experimental evaluation showed that the Random Forest model, when trained on the reduced feature set, yielded the highest accuracy (87%), making it the optimal model for this classification task. This result highlighted the effectiveness of feature selection and dimensionality reduction, as both processes significantly improved the performance of the models. Additionally, the PCA visualization demonstrated that the benchmark classes were well-separated in the reduced feature space, facilitating effective classification even with minimal features.

The performance evaluation through standard accuracy metrics confirmed that machine learning models could accurately categorize the benchmarks, offering a promising solution for future benchmarking and profiling tasks in similar contexts. This classification framework could serve as a foundation for efficient resource allocation, system optimization, and performance analysis.

While the results of this thesis demonstrate promising performance in benchmark classification, there are several directions in which this research can be expanded and improved. Some potential avenues for future work include:

- **Exploring Additional Models**: Although Random Forest performed the best in this study, other machine learning models, such as deep learning techniques or support vector machines, could be explored for further performance improvements. Additionally, ensemble methods combining multiple models may lead to better accuracy and robustness.
- **Handling Larger Datasets**: As the dataset size increases, the ability of machine learning models to generalize and perform effectively will become increasingly important. Further research could focus on extending this methodology to larger, more complex datasets.
- **Incorporating Other Profiling Metrics**: The current approach relied primarily on memory bandwidth and CPU utilization. Future work could include additional profiling metrics such

as cache hit ratios, latency, or power consumption, which could provide more comprehensive insights and improve classification accuracy.

By addressing these future challenges, the approach presented in this thesis could evolve into a highly effective tool for benchmarking and performance analysis in diverse computing environments, offering both predictive capabilities and actionable insights for system optimization.

Bibliography

[1] Z. Li and M. Wu, "Transparent and lightweight object placement for managed workloads atop hybrid memories," in *Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2022, pp. 72–80.

[2] S. Akram, "Performance evaluation of intel optane memory for managed workloads," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 3, pp. 1–26, 2021.

[3] M. Jordà, S. Rai, E. Ayguadé, J. Labarta, and A. J. Peña, "Ecohmem: Improving object placement methodology for hybrid memory systems in hpc," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, 2022, pp. 278–288.

[4] D. Shen, X. Liu, and F. X. Lin, "Characterizing emerging heterogeneous memory," *ACM SIGPLAN Notices*, vol. 51, no. 11, pp. 13–23, 2016.

[5] M. S. Marques, "Pnp: Rethinking page placement and persistence mechanisms to support the new intel optane dc persistent memory,"

[6] M. Marques, "Ambix: Rethinking linux's page management to support the new intel optane dc persistent memory," *Memory*, vol. 1, p. L3, 2021.

[7] S. Kumar, A. Prasad, S. R. Sarangi, and S. Subramoney, "Radiant: Efficient page table management for tiered memory systems," in *Proceedings of the 2021 ACM SIGPLAN International Symposium on Memory Management*, 2021, pp. 66–79.

[8] J. Kim, W. Choe, and J. Ahn, "Exploring the design space of page management for {multi-tiered} memory systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 715–728.

[9] J. Izraelevitz et al., "Basic performance measurements of the intel optane dc persistent memory module," *arXiv preprint arXiv:1903.05714*, 2019.

[10] H. Bahn and K. Cho, "Implications of nvm based storage on memory subsystem management," *Applied Sciences*, vol. 10, no. 3, p. 999, 2020.

[11] P. Zuo, Y. Hua, and J. Wu, "{Write-optimized} and {high-performance} hashing index scheme for persistent memory," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 461–476.

[12] K. Wu, Y. Huang, and D. Li, "Unimem: Runtime data managementon non-volatile memory-based heterogeneous main memory," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–14.

[13] J. Yang, J. Kim, M. Hoseinzadeh, J. Izraelevitz, and S. Swanson, "An empirical guide to the behavior and use of scalable persistent memory," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 169–182.

[14] C. Cantalupo, V. Venkatesan, J. Hammond, K. Czurlyo, and S. D. Hammond, "Memkind: An extensible heap memory manager for heterogeneous memory platforms and mixed memory policies.," Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2015.

[15] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), *ISO/IEC 9899:2024 - Information technology – Programming languages – C*. ISO/IEC, 2024.

[16] S. Shirvankar, "Statichmem: Static object placement methodology for heterogeneous memory systems in hpc," M.S. thesis, Universitat Politècnica de Catalunya, 2023.

[17] Memkind(3) API [Online] [Accessed: 2025].

[18] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, pp. 81–106, 1986.
[19] V. Podgorelec, P. Kokol, B. Stiglic, and I. Rozman, "Decision trees: An overview and their use in medicine," *Journal of medical systems*, vol. 26, pp. 445–463, 2002.
[20] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.
[21] A. Cutler, D. R. Cutler, and J. R. Stevens, "Random forests," *Ensemble machine learning: Methods and applications*, pp. 157–175, 2012.
[22] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, pp. 197–227, 2016.
[23] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, "Variable selection using random forests," *Pattern recognition letters*, vol. 31, no. 14, pp. 2225–2236, 2010.
[24] J. Laaksonen and E. Oja, "Classification with learning k-nearest neighbors," in *Proceedings of international conference on neural networks (ICNN'96)*, IEEE, vol. 3, 1996, pp. 1480–1483.
[25] G. Batista, D. F. Silva, et al., "How k-nearest neighbor parameters affect its performance," in *Argentine symposium on artificial intelligence*, Citeseer, 2009, pp. 1–12.
[26] P. Cunningham and S. J. Delany, "K-nearest neighbour classifiers-a tutorial," *ACM computing surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021.
[27] J. Joyce, "Bayes' theorem," 2003.
[28] F.-J. Yang, "An implementation of naive bayes classifier," in *2018 International conference on computational science and computational intelligence (CSCI)*, IEEE, 2018, pp. 301–306.
[29] Y. Huang and L. Li, "Naive bayes classification algorithm based on small sample set," in *2011 IEEE International conference on cloud computing and intelligence systems*, IEEE, 2011, pp. 34–39.
[30] I. Rish et al., "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Seattle, WA, USA; vol. 3, 2001, pp. 41–46.
[31] K. P. Murphy et al., "Naive bayes classifiers," *University of British Columbia*, vol. 18, no. 60, pp. 1–8, 2006.
[32] S. Vishwanathan and M. N. Murty, "Ssvm: A simple svm algorithm," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, IEEE, vol. 3, 2002, pp. 2393–2398.
[33] A. Patle and D. S. Chouhan, "Svm kernel functions for classification," in *2013 International conference on advances in technology and engineering (ICATE)*, IEEE, 2013, pp. 1–9.
[34] V. Jakkula, "Tutorial on support vector machine (svm)," *School of EECS, Washington State University*, vol. 37, no. 2.5, p. 3, 2006.
[35] A. Mathur and G. M. Foody, "Multiclass and binary svm classification: Implications for training and classification users," *IEEE Geoscience and remote sensing letters*, vol. 5, no. 2, pp. 241–245, 2008.
[36] S. Karamizadeh, S. M. Abdullah, A. A. Manaf, M. Zamani, and A. Hooman, "An overview of principal component analysis," *Journal of signal and information processing*, vol. 4, no. 3, pp. 173–175, 2013.
[37] G. Ivosev, L. Burton, and R. Bonner, "Dimensionality reduction and visualization in principal component analysis," *Analytical chemistry*, vol. 80, no. 13, pp. 4933–4944, 2008.
[38] P. M. Shenai, Z. Xu, and Y. Zhao, "Applications of principal component analysis (pca) in materials science," *Princ. Compon. Anal. Appl*, pp. 25–40, 2012.
[39] M. Mavungu, "Computation of financial risk using principal component analysis," *Algorithmic Finance*, vol. 10, no. 1-2, pp. 1–20, 2023.
[40] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.
[41] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*, Ieee, 2009, pp. 44–54.
[42] J. Poovey et al., "Characterization of the eembc benchmark suite," *North Carolina State University*, vol. 32, pp. 37–50, 2007.
[43] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.

[44] Y. S. U. Vishkin and Y. Shiloach, "An o (log n) parallel connectivity algorithm," *J. algorithms*, vol. 3, pp. 57–67, 1982.