# קידוד ואלגוריתמים לזכרונות 236379
# חורף תשפ״א (2020-2021)

מגישים:

| שם וכתובת דוא״ל | תעודת זהות |
|---|---|
| אייל לוטן – eyallotan@campus.technion.ac.il | 302288527 |
| דור סורה – dorsura@campus.technion.ac.il | 204098784 |

**מרצה: פרופ׳. חבר איתן יעקובי**

**מתרגלת: גב. דניאלה בר לב**

# Garbage Collection Algorithms for Flash Memories

## הקדמה

בפרויקט זה נעסוק באלגוריתמי GC עבור זכרונות פלאש (SSD). נציג מודל תאורטי אשר יאפשר לנו לפתח מספר שיפורים ואלגוריתמים לניהול זיכרון יעיל יותר, בהשוואה לאלגוריתמים הקלאסיים הנמצאים בשימוש כיום. כל התוצאות בפרויקט זה מבוססות על ניסויים שהורצו על סימולטור בו מומשו כלל האלגוריתמים והשיפורים אשר יוצגו בהמשך. כל הקוד הרלוונטי להרצת הסימולטור נמצא ב-github. תחת הקישור ניתן למצוא את כל ההוראות להורדה והפעלה של הסימולטור. לשם הפשטות, בתיאורים האלגוריתמיים המובאים בדו"ח זה יוצגו האלגוריתמים בצורה כללית, מבלי להיכנס לכל פרטי המימוש. כל התיעוד הרלוונטי למימוש האלגוריתמים מצורף לקוד הסימולטור.

## מבוא והגדרות

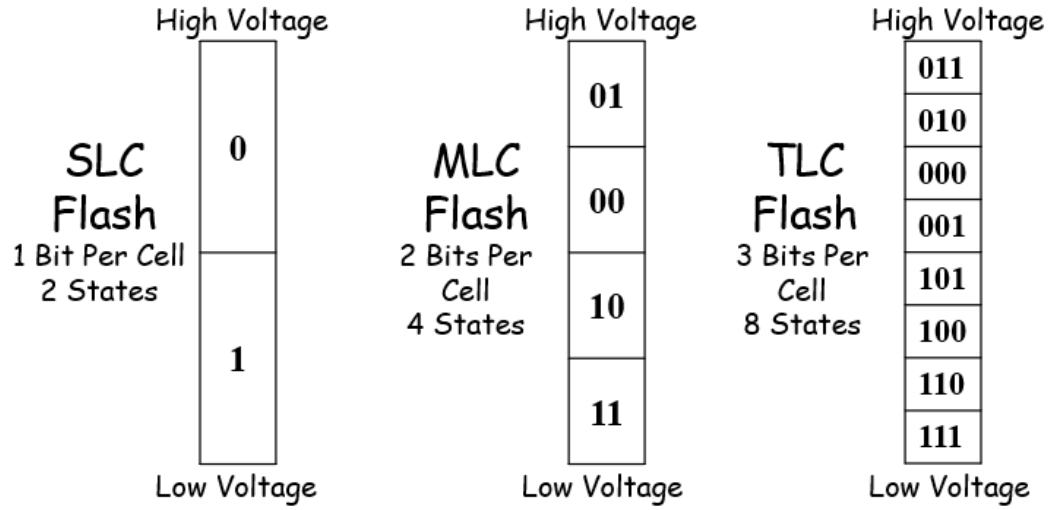
נציג תחילה את המודל התיאורטי בו נעסוק, וכמו כן נציין מספר הגדרות אשר ישמשו אותנו בפרקים הבאים.

זיכרון פלאש הינו רכיב זיכרון אלקטרוני מסוג non-volatile. הזיכרון מאפשר מחיקה וכתיבה מחדש (reprogramming) של מידע. זכרונות פלאש נמצאים בשימוש במחשבים, פלאפונים, מצלמות ומכשירים אלקטרוניים רבים נוספים.

### מבנה וארגון הזכרון

זכרונות פלאש שומרים מידע במערך של תאי זכרון הבנויים מטרנזיסטורים מסוג floating gate. ישנם כמה סוגים של תאים:

- Single level cell (SLC) – כל תא מכיל ביט אחד של אינפורמציה. התא יכול להיות באחד משני מצבים (0 או 1).
- Multi level cell (MLC) – כל תא מכיל 2 ביטים של אינפורמציה (עד ארבעה מצבים שונים).
- Triple level cell (TLC) – כל תא מכיל 3 ביטים של אינפורמציה (עד 8 מצבים שונים).




איור 1: מבנה תא זכרון.

קבוצה של תאי זיכרון מרכיבה דף זיכרון (page). קבוצה של דפים מרכיבה בלוק (block). הזיכרון כולו בנוי מאוסף של בלוקים.

הפעולות שניתן לבצע על הזיכרון:

1. קריאה/כתיבה של דף בודד. כאשר אנחנו כותבים דף לזיכרון, נאמר שהדף הפיזי בזיכרון נמצא במצב valid. סטטוס זה מסמל שהדף "חי" ומכיל מידע עדכני, וניתן לגשת אליו לצורך קריאה.
2. מחיקה של בלוק שלם.

נשים לב כי תחת הגדרות אלו, המגבלה העיקרית העומדת בפנינו היא **שאין אפשרות לבצע כתיבה חוזרת לאותו דף (rewrite) מבלי למחוק את הבלוק כולו**. לכן, בכל פעם שנרצה לשנות את התוכן של דף כלשהו, ניאלץ לכתוב אותו מחדש, ולסמן את המיקום הישן שלו עם ערך מיוחד אשר יסמן לנו שהדף נמצא במצב invalid ולא ניתן לכתוב אליו. לאחר זמן מה הזיכרון עלול להתמלא, כאשר חלק מהדפים הם במצב valid ואילו אחרים במצב invalid. כעת, על מנת לאפשר כתיבה של דפים חדשים, עלינו לבצע תהליך איסוף זבל (Garbage Collection – GC) אשר עובד בשיטת copy-erase-write:

1. אלגוריתם פינוי הבלוקים בוחר בלוק לפינוי.
2. כל הדפים אשר נמצאים בסטטוס valid בבלוק מועתקים ל-buffer זמני.
3. מבצעים מחיקה של הבלוק.
4. הדפים ה-valid נכתבים בחזרה אל הבלוק בצורה סדרתית מתחילתו.

בצורה זו ניתן להמשיך ולכתוב דפים חדשים אל הבלוק.

בשל מגבלות הזיכרון שהוזכרו לעיל, עלינו לתחזק בלוקים נוספים (מעבר לאלו המובטחים למשתמש) על מנת לאפשר להמשיך ולכתוב דפים מבלי לעצור ולבצע פינוי של בלוקים לעיתים תכופות. לשם כך נעשה הפרדה בין הבלוקים הלוגיים המובטחים למשתמש לבין הבלוקים הפיזיים.

<u>הגדרות וסימונים</u>

| סימון | הגדרה |
|---|---|
| $T$ | מספר הבלוקים הפיזיים |
| $U$ | מספר הבלוקים הלוגיים |
| $Z$ | מספר הדפים בבלוק |
| $N$ | מספר הכתיבות של דפים לוגיים |
| $M$ | מספר הכתיבות של דפים פיזיים |
| $E$ | מספר מחיקות בלוקים |

נתייחס לדפים הלוגיים כאל הדפים בטווח $[0, U \cdot Z - 1]$.

מכאן נוכל להגדיר את המושג over-provisioning (OP) – זהו היחס בין כמות הזיכרון הפיזי הנוסף לבין הזיכרון הלוגי. זוהי היתירות הדרושה לצורך ביצוע כתיבות חוזרות של דפים (out of place writes). מתקיים: $OP = \frac{T-U}{U}$. בנוסף נגדיר גם את ה-storage rate להיות $\alpha = \frac{U}{T}$.

מכיוון שכתיבות של דפים מתבצעות out of place, עלינו לתחזק מיפוי אשר ממפה בין דפים לוגיים לדפים פיזיים. לשם כך קיימת שכבת ה-Flash Translation Layer (FTL) אשר אחראית על שמירה ותחזוקת המיפוי בין דפים לוגיים לפיזיים. ה-FTL הינו גם חלק מזיכרון הפלאש, ובכל פעם שהמתח מנותק יש לוודא שמירה מאובטחת של המיפוי על מנת לוודא שהמידע שנכתב יישאר מעודכן גם לאחר הפעלה מחדש.

דוגמה ל-memory layout של זכרון פלאש:

נתוני הזיכרון – $T = 5, U = 4, Z = 4$.

| Block number 1 | Block number 2 | Block number 3 | Block number 4 | Block number 5 |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

המשתמש יכול לכתוב דפים לוגיים בטווח [0,15]. הדפים ייכתבו לבלוק פנוי באופן סדרתי, והמיפויים הרלוונטיים יישמרו ב-FTL.

מצב הזיכרון לאחר סדרה של 8 כתיבות:

| Block number 1 | Block number 2 | Block number 3 | Block number 4 | Block number 5 |
|---|---|---|---|---|
| Page 3 | Page 11 | | | |
| Page 0 | Page 8 | | | |
| Page 6 | Page 7 | | | |
| Page 12 | Page 10 | | | |

נשים לב כי אין בהכרח קשר בין מספר הדף הלוגי של הדף הנכתב לבין מספר הדף הפיזי אליו הדף ייכתב בפועל. הדפים בפועל נכתבים בצורה סדרתית ושכבת ה-FTL היא זו שדואגת לשמור על המיפוי התקין.

כעת נניח כי מגיעה בקשת לכתיבת דף לוגי מס' 7. נזהה כי הדף הלוגי כבר ממופה לדף פיזי, ולכן נסמן את הדף הנוכחי כ-invalid ונכתוב את הדף למקום הפנוי הבא. לאחר הכתיבה הזיכרון ייראה כך:

| Block number 1 | Block number 2 | Block number 3 | Block number 4 | Block number 5 |
|---|---|---|---|---|
| Page 3 | Page 11 | Page 7 | | |
| Page 0 | Page 8 | | | |
| Page 6 | Invalid | | | |
| Page 12 | Page 10 | | | |

מצב הזיכרון לאחר סדרה של כתיבות המביאה למילוי הזיכרון הפיזי:

| Block number 1 | Block number 2 | Block number 3 | Block number 4 | Block number 5 |
|---|---|---|---|---|
| Page 3 | Page 11 | Invalid | Page 2 | Page 8 |
| Invalid | Invalid | Page 1 | Page 5 | Page 10 |
| Page 6 | Invalid | Page 0 | Page 7 | Invalid |
| Page 12 | Page 10 | Page 4 | Invalid | Page 13 |

כעת נניח שמגיעה בקשה לכתיבה של דף לוגי מס' 5. תחילה נשתמש בשכבת ה-FTL ונגלה שהדף קיים בזיכרון הפיזי (בבלוק מס' 4), ולכן נסמן את המיקום הישן של הדף בתור invalid. כאשר ננסה לכתוב את הדף לזיכרון נגלה כי אין דפים פיזיים פנויים, ולכן אלגוריתם ה-GC ייכנס לפעולה ויבחר בלוק למחיקה. נניח כי בלוק מס' 2 נבחר למחיקה. בבלוק זה קיימים שני דפים עם סטטוס Valid – דף מס' 11 ודף מס' 10. שני הדפים הללו יועתקו ל-buffer זמני, ולאחר מכן נוכל למחוק את הבלוק לגמרי. לבסוף נכתוב בחזרה את שני הדפים ה-valid בחזרה לבלוק מס' 2, ואחריהם נוכל לכתוב את דף מס' 5. מצב הזיכרון לאחר GC וכתיבה של דף מס' 5:

| Block number 1 | Block number 2 | Block number 3 | Block number 4 | Block number 5 |
|---|---|---|---|---|
| Page 3 | Page 11 | Invalid | Page 2 | Page 8 |
| Invalid | Page 10 | Page 1 | Invalid | Page 10 |
| Page 6 | Page 5 | Page 0 | Page 7 | Invalid |
| Page 12 | | Page 4 | Invalid | Page 13 |

בנוסף, עבור דוגמה זו מתקיים: $OP = \frac{T-U}{U} = \frac{5-4}{4} = 0.25$.

כעת נגדיר:

| סימון | הגדרה |
|---|---|
| $P$ | מספר הכתיבות לדפים פיזיים |
| $L$ | מספר הבקשות לכתיבת דפים לוגיים |

Write Amplification (WA) – היחס בין מס' הכתיבות הפיזיות לדפים בזיכרון לבין מס' הבקשות לכתיבת דפים לוגיים. מתקיים: $WA = \frac{P}{L}$. מדד זה מתאר את המחיר שעלינו לשלם בעבור מחיקה של בלוקים ותהליך ה-copy-erase-write שתואר לעיל.

Erasure Factor (EF) – היחס בין מספר מחיקות הבלוקים לבין מספר הכתיבות הלוגיות של הבלוקים. מתקיים: $EF = \frac{E}{L/Z}$.

בהנחות העבודה שלנו בפרויקט זה נניח כי אין שימוש בכתיבות חוזרות לבלוקים. תחת הנחות אלו מתקיים כי $WA = EF$.

<u>התפלגות הכתיבות:</u>

עבור בקשת כתיבה של דף לוגי כלשהו מתוך הדפים בטווח $[0, U \cdot Z - 1]$, נתייחס לשתי התפלגויות כתיבה אפשריות:

1. Uniform Writing – הדף לכתיבה נבחר יוניפורמית על פני כל הדפים הלוגיים.
2. Hot-Cold Writing – התפלגות כתיבה זו מדמה מצב בו יש לנו זיכרון המחולק לאזור של דפים חמים ואזור של דפים קרים. כפי ששמם מרמז, דפים חמים אלו דפים אשר נכתבים בתדירות גבוהה יותר. לרוב מדובר בקבוצה קטנה של דפים "נעוצים" (pinned memory pages). מנגנון זה יכול גם לעזור לדמות מצב של שימוש ב-cache. על מנת להגדיר מודל כתיבה זה נשתמש בשני פרמטרים נוספים:

   $r$ – החלק היחסי של הדפים החמים מתוך כלל הדפים הלוגיים.

   $p$ – ההסתברות לכתיבת דף חם.

## מטרת הפרויקט

לזכרונות פלאש ישנה תוחלת חיים התלויה במספר המחיקות המתבצעות לכל בלוק בזיכרון. אחד המדדים המשתמשים כדי לאמוד את הבלאי של הזיכרון (memory wear) הוא P/E cycles (program erase cycles). מחזור P/E הוא רצף אירועים בו מידע נכתב לזיכרון, נמחק ואז נכתב מחדש. לכל רכיב פלאש ישנו מספר סופי של מחזורי P/E אשר הוא יכול לבצע עד שמתחיל להיווצר בלאי לרכיבים הפיזיים של תאי הזיכרון, אשר עשויים לגרום לרכיב להפוך ללא שמיש. "תוחלת החיים" של ה-SSD, או מס' מחזורי ה-P/E אשר ניתן לבצע עד לבלאי בלתי הפיך של הרכיבים הפזיים, תלוי בסוג תאי הזיכרון, ויכול לנוע בין מאות מחזורי P/E עבור תאי TLC ועד לעשרות אלפי מחזורי כתיבה עבור תאי SLC. המספרים יכולים לנוע כתלות ביצרן ובטכנולוגיות נוספות אשר מתווספות לדיסק במטרה להאריך את חייו (כמו קודים לתיקון שגיאות לדוגמה). בטבלה מצורפת הערה של מספר מחזורי P/E לפי טכנולוגיית התאים שבשימוש. המספרים המצוינים מהווים הערכה גסה ומטרתם לתת קנה מידה לגבי הבעיה איתה אנחנו מתמודדים:

| Cell Type | P/E cycles |
|---|---|
| SLC | $50000 - 100000$ |
| MLC | $3000 - 10000$ |
| TLC | $300 - 1000$ |

בפרויקט זה נרצה להתמקד בדרכים בהם נוכל לצמצם את מס' מחיקות הבלוקים עבור סדרת כתיבות נתונה. כלומר, המדד אותו ננסה למזער הינו ה-write amplification, או באופן שקול ה- erasure factor.

## Greedy GC

נציג בקצרה את האלגוריתם Greedy GC שנלמד במסגרת הקורס. **עפ"י משפט, תחת ההנחה כי התפלגות הכתיבות יוניפורמית, אלגוריתם זה הינו אופטימלי וישיג את ה-WA המינימלי.**

נגדיר תחילה פרמטר נוסף ומבני נתונים אשר ישמשו אותנו בתיאור אלגוריתם זה ואלגוריתמים נוספים בהמשך: נסמן ב-$Y$ את ערך ה-minValid. בהינתן כל הבלוקים הפיזיים, ערך ה-minValid מתאר את המס' המינימלי של דפים ולידיים בבלוק פיזי כלשהו בזיכרון. בממוצע (תחת הנחת הפיזור האחיד), מתקיים כי $Y = \alpha' Z$, כאשר $\alpha' = \frac{EF-1}{EF} = \frac{EZ-L}{EZ}$.

נגדיר את ה-$freelist$ להיות רשימה מקושרת של בלוקים המכילה את הבלוקים בהם יש מקומות פנויים לכתיבה. אם ה-$freelist$ מכילה לפחות בלוק אחד – נוכל לכתוב את הדפים הבאים לבלוק שבראש הרשימה המקושרת. ברגע שהבלוק בראש הרשימה מלא, הוא מוסר מהרשימה. בזמן האתחול כל הבלוקים מתווספים ל-$freelist$.

נגדיר את $V$ להיות מערך של מצביעים בגודל $Z + 1$. לכל $0 \leq i \leq Z$ נגדיר את $V[i]$ להיות קבוצה המוגדרת באופן הבא:

$$V[i] = \{\, u \in [0, U-1] \mid block\ u\ is\ full\ and\ has\ i\ valid\ pages \}$$

כלומר, כל איבר במערך מכיל את כל הבלוקים המלאים אשר לכולם אותו מספר של דפים ולידיים.

תיאור אלגוריתם הכתיבה:

Given a logical page $lpn \in [0, U \cdot Z - 1]$:

1. $if\ freelist\ is\ not\ empty$:
   1.1. $block = freelist \rightarrow front$.
   1.2. Write $lpn$ to $block$.
   1.3. Update FTL and return.
2. else, call $GC$ and goto 1.

תיאור אלגוריתם ה-$GC$:

1. Compute $Y$.
2. Let $minValidSet = V[Y]$.
3. Pick at random a block $B$ from $minValidSet$ and perform a copy-erase-write operation. Update the FTL accordingly.
4. Add $B$ to $freelist$.

אלגוריתם זה ישמש כנקודת מוצא להשוואה עבור כל השיפורים והאלגוריתמים שיוצגו בהמשך.

כאמור, תחת ההנחות שהוצגו עד כה, אלגוריתם Greedy GC הינו אופטימלי. על מנת להצליח ולשפר את ביצועיו, נציג כעת את מודל וההנחות לפיהן נפתח את האלגוריתמים שלנו.

**<u>הצגת המודל וההנחות</u>**

הנחת העבודה שלנו עד כה הייתה שאין לנו מידע לגבי בקשות הכתיבה, מעבר לידע על האופן בו הן מתפלגות. נניח כעת כי סדרת הכתיבות ידועה לנו מראש.

פורמלית – בהינתן מספר הדפים לכתיבה $N$, נניח כי נתונה לנו סדרה של כתיבות $writing\_sequence = \{ lp_1, lp_2, \cdots, lp_N\}$. כאשר $lp_i \in [0, U \cdot Z - 1]$ לכל $1 \leq i \leq N$. עלינו לכתוב את כל הדפים לפי הסדר בו הם מופיעים ב-$writing\_sequence$ (בקיצור $ws$). לאחר קבלת סדרת הכתיבות, לא ניתן לשנות את סדר הכתיבות ו/או להשמיט כתיבות של דפים. בנוסף, מובטח כי סדרת הכתיבות נוצרה ע"י בחירת דפים בצורה יוניפורמית (בהמשך נתייחס גם למקרה בו התפלגות הכתיבות היא Hot-Cold).

**מטרתנו היא לתכנן אלגוריתמים לכתיבה של סדרת הדפים $ws$ לזיכרון, באופן שיביא למזעור ה-write amplification, בהשוואה לכתיבה של אותה סדרת הדפים באמצעות אלגוריתם Greedy GC** (כאשר עבור Greedy הדפים נכתבים לפי האלגוריתם שפורט לעיל).

<u>Steady state assumption</u>

בכל הניסויים והסימולציות שנבצע נניח כי בטרם הכתיבה של סדרת הכתיבות המבוקשת, הזיכרון מגיע למצב של steady state – כלומר הזיכרון מלא בדפים אשר נכתבו בצורה יוניפורמית. לצורך הבאה של הזיכרון למצב steady state נשתמש באלגוריתם Greedy GC לצורך פינוי בלוקים. הנחה זו מקלה על החישובים וחוסכת מאיתנו את ההתמודדות עם סדרת הכתיבות הראשונות אשר נכתבות לזיכרון כאשר הוא עדיין ריק. כמובן שנשתמש ב-steady state assumption גם עבור הרצת אלגוריתם ה-Greedy אליו אנחנו משווים את האלגוריתמים שלנו. מספר הכתיבות הדרוש עד להגעה למצב steady state משתנה ותלוי במספר הבלוקים שנבחרו לסימולציה. ניתן לשנות פרמטר זה כרצונכם (פרטים נוספים בקובץ README.md).

הנחות נוספות:

1. נניח כי גודל כתיבה היא תמיד של דף בודד. כלומר כל כתיבה יכולה להיכנס בתוך בלוק בודד (בהנחה ובבלוק ישנם דפים פנויים).
2. אין הכרח שיהיו בלוקים אשר תמיד נשארים ריקים. הנחה זו קיימת לעיתים בתוך זכרונות SSD על מנת שלא לעכב כתיבות גדולות אשר עלולות "להיתקע" בשל המתנה ל-GC. מכיוון שכתיבה היא תמיד של דף בודד, אין לנו צורך לשמור בלוקים פנויים עבור כתיבות גדולות, ובכך אנחנו מרוויחים ניצולת מקסימלית של הזיכרון הפיזי וממזערים את ה-WA אשר בו אנחנו רוצים להתמקד בפרויקט זה.
3. נניח כי קיים temporary buffer בזיכרון אשר משמש להעתקה של דפים ולידיים עבור בלוקים שעוברים GC.
4. נניח כי כל ה-metadata של הבלוקים וה-FTL לא נשמר בתוך הבלוקים עצמם אלא בזיכרון, ולכן כל שינוי ל-FTL או למידע הנוסף השמור עבור כל בלוק אינו נספר ככתיבה ולא משפיע על ה-WA.

נציג כעת את השיפור הראשון לאלגוריתם Greedy GC הקלאסי.

## Greedy Lookahead GC

רעיון כללי ומוטיבציה

השיפור הראשון אותו נראה הינו למעשה אופטימיזציה לאלגוריתם Greedy GC. מטרתנו תהיה להסתכל על הנקודה בה מופעל אלגוריתם GC, ולבצע את הבחירה של הבלוק הבא לפינוי בצורה מושכלת, תוך שימוש בידע שיש לנו על הכתיבות העתידיות לבוא.

בשלב 2 של אלגוריתם Greedy GC אנחנו למעשה מסתכלים על הבלוקים בקבוצה $V[Y]$ ובוחרים בלוק **כלשהו** אשר יהיה "קורבן" עבור ה-GC.

כעת נרצה לשאול את השאלה – בהינתן $V[Y]$ ו-$ws$ המכיל את הכתיבות העתידיות, האם נוכל לשפר את האופן בו אנחנו בוחרים את הבלוק הבא לפינוי?

נמחיש זאת בעזרת דוגמה – נניח כי הגיעה כתיבה אשר גרמה להפעלת אלגוריתם ה-GC, ומתקיים כי $Y = 3$ ובנוסף $V[Y] = \{1,3,5\}$. נסתכל על הבלוקים בקבוצה $V[Y]$ ונסתכל על מספרי הדפים הלוגיים הולידיים בכל בלוק (נניח כמובן כי כל מספרי הדפים הם מספרים בטווח החוקי של הדפים הלוגיים):

| Block no. 1 | Block no. 3 | Block no. 5 |
|---|---|---|
| 1 | X | 3 |
| X | X | X |
| 4 | 2 | 5 |
| X | X | X |
| 6 | X | 7 |
| X | 9 | X |
| X | 8 | X |

ונניח כי נתון לנו מקטע הכתיבה הבא כחלק מה-$ws$ (הכתיבה הראשונה שמופיעה היא הכתיבה ה-$i$ שגרמה ל-GC): $ws = [\cdots,1,6,4,3,9,\cdots]$. אלגוריתם ה-Greedy GC הנאיבי יבחר באופן שרירותי את אחד מהבלוקים בקבוצה $V[Y]$ ויבצע עליו מחיקה וכתיבה מחדש של הדפים הולידיים. מנגנון הבחירה תלוי מימוש כמובן, אך בכל מקרה ניתן להניח שהבחירה תהיה שקולה לבחירה שרירותית כלשהי. גם כאן ניתן לסייג ולומר כי אלגוריתם ה-Greedy יכול לשמור תיעוד של אילו דפים נכתבו עד כה, ומכך להסיק לאילו דפים יש יותר סיכוי להיכתב בעתיד הקרוב (בהתבסס על הנחת הפיזור האחיד), אך לא נעמיק בסוגיה זו כרגע (וזהו גם מנגנון שלא קיים כחלק מאלגוריתם Greedy GC המקורי).

כעת נניח כי אלגוריתם Greedy GC בוחר למחיקה את בלוק מס' 1. לאחר המחיקה והכתיבה של הדפים הולידיים נקבל:

| Block no. 1 | Block no. 3 | Block no. 5 |
|---|---|---|
| 1 | X | 3 |
| 4 | X | X |
| 6 | 2 | 5 |
| | X | X |
| | X | 7 |
| | 9 | X |
| | 8 | X |

כעת, נמשיך לבצע את רצף הכתיבות לפי ה-$ws$ הנתון לעיל. תמונת הזיכרון לאחר ביצוע ארבעת הכתיבות הבאות:

| Block no. 1 | Block no. 3 | Block no. 5 |
|---|---|---|
| X | X | X |
| X | X | X |
| X | 2 | 5 |
| 1 | X | X |
| 6 | X | 7 |
| 4 | 9 | X |
| 3 | 8 | X |

כעת, כשנגיע לכתוב את דף מס' 9, נראה כי לא נשאר מקום פנוי לכתיבה ולכן אלגוריתם ה-GC שוב ייכנס לפעולה. מכיוון שכל הכתיבות שביצענו "נגעו" רק בבלוקים הנתונים לעיל, לא ייתכן שדפים בבלוקים אחרים בזיכרון סומנו כ-invalid. מכאן, בהכרח נקבל כי $Y = 2$ ו- $V[Y] = \{5\}$. כעת, עבור הכתיבה של דף לוגי מס' 9 אנחנו משלמים "קנס" של 2 כתיבות פיזיות נוספות – הכתיבות שמקורן בהעתקה של 2 דפים ולידיים לאחר מחיקת בלוק מס' 5.

כעת נחזור על התהליך, רק שבמקום לבחור את בלוק מס' 1 למחיקה, נבחר את בלוק מס' 3. לאחר המחיקה והכתיבה של הדפים הולידיים נקבל:

| Block no. 1 | Block no. 3 | Block no. 5 |
|---|---|---|
| 1 | 2 | 3 |
| X | 9 | X |
| 4 | 8 | 5 |
| X | | X |
| 6 | | 7 |
| X | | X |
| X | | X |

כעת, נמשיך לבצע את רצף הכתיבות לפי ה-$ws$ הנתון. תמונת הזיכרון לאחר ביצוע ארבעת הכתיבות הבאות ברצף:

| Block no. 1 | Block no. 3 | Block no. 5 |
|---|---|---|
| X | 2 | X |
| X | 9 | X |
| X | 8 | 5 |
| X | 1 | X |
| X | 6 | 7 |
| X | 4 | X |
| X | 3 | X |

כעת, כשנגיע לכתוב את דף מס' 9, נשים לב כי מתקיים $Y = 0$ ו-$V[Y] = \{1\}$. כלומר, אנחנו לא משלמים penalty כלל על כתיבת הדף הבא. נוכל למעשה למחוק את בלוק מס' 1 מבלי להעתיק אף דף נוסף, וישר לאחר מכן לבצע אליו כתיבה מחדש של הדף המבוקש.

מדוגמה זו נסיק מסקנה טריוויאלית אך מעניינת:

<u>מסקנה 1</u>: ככל שהבלוקים אותם אנחנו מוחקים מכילים פחות דפים ולידיים (כלומר ערך ה-$Y$ עבורם מינימלי), כך אנחנו מקטינים את מס' הכתיבות הפיזיות הכולל ובכך גם את ה-write amplification.

ננסה למדל את ההתנהגות של הזיכרון בדוגמה לעיל – למעשה, אם נבחר את בלוק מס' 1 למחיקה הראשונה, אנחנו מעתיקים שלושה דפים אשר **בעתיד הקרוב** בלאו הכי הולכים להפוך להיות invalid. לכן, אם ידוע לנו שהדפים האלו הולכים להיכתב מחדש, לא נרצה לבזבז את המשאבים שלנו בהעתקה שלהם. יתרה מכך, בכך שנשאיר את הדפים בבלוק המקורי שלהם, באופן אפקטיבי נקטין את $Y$, וכך לפי מסקנה 1 נצליח להשיג את ההתנהגות הרצויה המביאה להקטנת ה-write amplification.

<u>מסקנה 2</u>: הבלוק אשר נרצה לבחור למחיקה הוא הבלוק שמכיל את הדפים להם ישנה "תוחלת החיים" הארוכה ביותר.

נשים לב שישנם שני נושאים חשובים אשר נידרש להתייחס אליהם בהמשך:

1. "תוחלת חיים" ו-עתיד – עלינו להגדיר במדויק את מושג ה-"עתיד הקרוב" אותו הגדרנו לעיל, ובנוסף את הדרך בה נמדוד את תוחלת החיים של דף נתון.

2. גרנולריות – יש לזכור כי אנחנו עובדים בגרנולריות של בלוקים. דהיינו, עלינו להתמודד עם סיטואציה בה בלוק מסוים מכיל גם דפים אשר עתידים להימחק בקרוב וגם דפים להם "תוחלת חיים" ארוכה יותר.

<u>תיאור האלגוריתם</u>

ניגש כעת לתיאור האלגוריתם Greedy Lookahead:

Suppose: $ws$ of size $N$, where $ws[i]$ is the i[th] logical page number to be written to memory.

Given a logical page $lpn_i \in [0, U \cdot Z - 1]$:

1. $if\ freelist\ is\ not\ empty$:
   1.1. $block = freelist \rightarrow front$.
   1.2. Write $lpn$ to $block$.
   1.3. Update FTL and return.
2. else, call $GCLookAhead$ and goto 1.

תיאור אלגוריתם ה-$GCLookAhead$:

1. Compute $Y$.
2. Let $minValidSet = V[Y]$.
3. foreach block $B$ in $minValidSet$:
   3.1. Calculate $block\_score(B, ws, i)$.
5. Choose $B = \underset{B \in V[Y]}{argmax}\{block\ score\}$, and perform a copy-erase-write operation on block $B$. Update the FTL accordingly.
4. Add $B$ to $freelist$.

פונקציית $block\_score$: פונקציה זו הינה הפונקציה בעזרתה ניתן הדירוג לכל אחד מהבלוקים. הפונקציה מוגדרת באופן הבא:

Define: $block\_score(B, ws, i)$:

1. Define $block\_score = 0$.
2. Define $pagesInBlock = \{p \in [0, U \cdot Z - 1] \mid p \in B\}$.
3. $for(long\ long\ pos = i\ ; pos < N\ ;\ pos++)$:
   3.1. if $ws[pos] \in pagesInBlock$:
      3.1.1. $pagesInBlock = pagesInBlock \setminus \{ws[pos]\}$.
      3.1.2. if $|pagesInBlock| = 0$ – return $block\_score$.
   3.2. $block\_score += |pagesInBlock|$.
4. Return $block\_score$.

הסבר – בהינתן בלוק $B$, הניקוד שיינתן לבלוק מבטא את "תוחלת החיים" של הדפים בו. נעשה זאת ע"י הסתכלות על כל הדפים הולידיים בתוך הבלוק $B$. נשמור את הדפים הללו בתוך הקבוצה $pagesInBlock$. עבור כל כתיבה עתידית – אם הכתיבה היא של דף הנמצא בבלוק $B$, נוציא את הדף מתוך $pagesInBlock$. לאחר מכן נוסיף לניקוד הבלוק את גודל הקבוצה $pagesInBlock$. בצורה זו, בלוק אשר הדפים בו נשארים ולידיים לאורך הרבה זמן יקבל ניקוד גבוה יותר וכך ייבחר קודם למחיקה.

מימוש האלגוריתם

אלגוריתם Greedy Lookahead ממומש בסימולטור. הוראות ההפעלה נמצאות בקובץ README.md.

תוצאות וניסויים

נרצה לבחון את ביצועי האלגוריתם החדש אל מול אלגוריתם Greedy GC המקורי.

הניסוי הראשון יבחן את השינוי ב-write amplification כתלות ב-over-provisioning. הפרמטרים לניסוי:

$$T = 64, Z = 32, N = 10^5$$

כל מדידה בניסוי היא של ערך write amplification. המדידה נקבעה על ידי לקיחת ממוצע של 20 הרצאות עבור הפרמטרים הרלוונטיים. נציג את התוצאות בטבלה:

| ALGORITHM / OP | GREEDY GC WA | GREEDY LOOKAHEAD WA |
|---|---|---|
| $U = 60, OP = 0.0666$ | 6.78036 | 6.65409 |
| $U = 56, OP = 0.1428$ | 3.81101 | 3.77215 |
| $U = 52, OP = 0.2307$ | 2.69367 | 2.67263 |
| $U = 48, OP = 0.3333$ | 2.11129 | 2.09632 |
| $U = 44, OP = 0.4545$ | 1.75364 | 1.74381 |
| $U = 40, OP = 0.6$ | 1.51706 | 1.50927 |

ניתן לראות כי אכן האלגוריתם החדש משיג שיפור בערכי ה-write amplification בצורה עקבית, לכל ערכי ה-OP השונים. תוצאה זו אינה מפתיעה – מטרתנו באופטימיזציית ה-Lookahead היא לבצע שיפורים **לוקליים**, אשר יבטיחו כי בכל הפעלה של ה-GC נבחר בלוק למחיקה שיהיה לכל הפחות טוב כמו הבחירה שהיה בוחר אלגוריתם Greedy GC הרגיל. בכך אנחנו מבטיחים כי לאורך זמן, הבחירות שיבצע אלגוריתם Lookahead יביאו לחיסכון בכמות המחיקות הכללי ושיפור ה-write amplification.

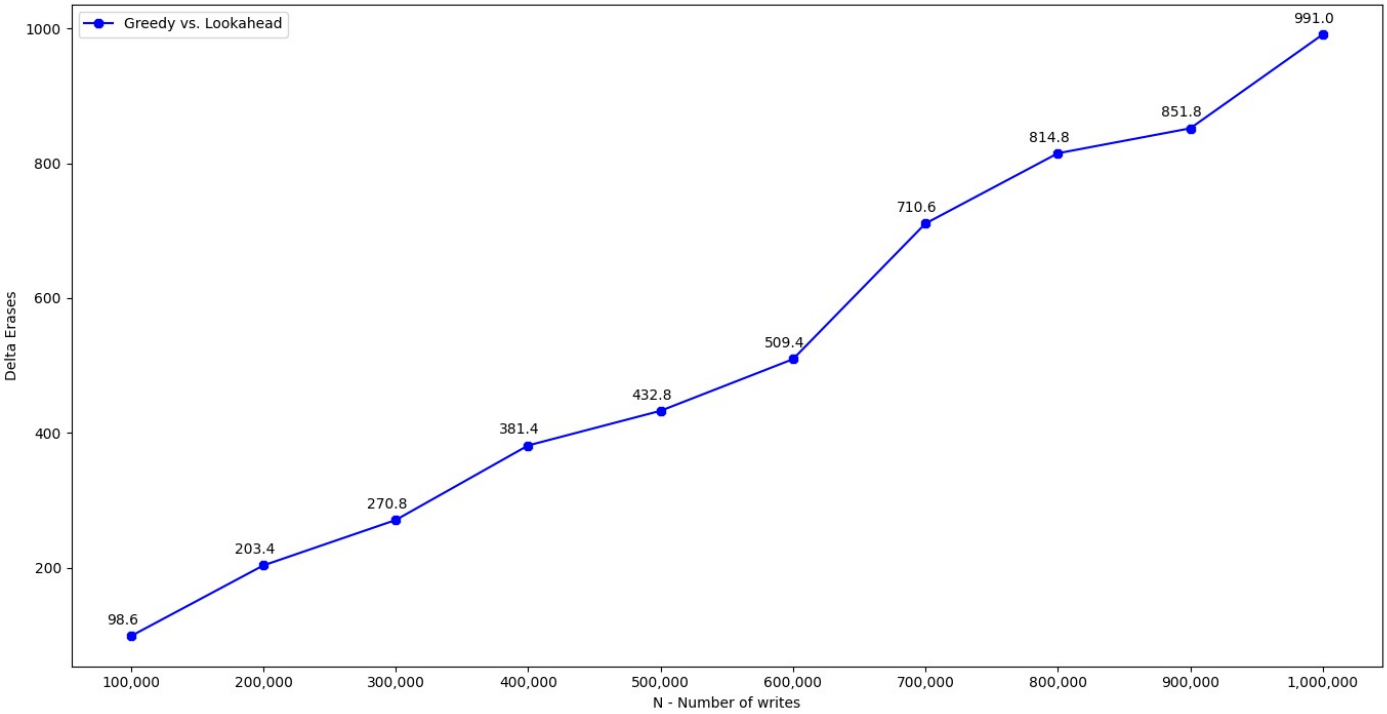
כדי לאשש את השערה זו, נרצה לבדוק האם החיסכון במספר המחיקות כתוצאה משימוש ב- Greedy Lookahead גדל ככל ש- $N$ גדל.

לצורך הניסוי הבא נגדיר את המדד הבא: $\Delta E = E_{Greedy} - E_{Lookahead}$. מדד זה מתאר את ההפרש בכמות המחיקות בין אלגוריתם Greedy GC לבין אלגוריתם Greedy Lookahead. בנוסף נגדיר את הפרמטרים הבאים עבור הניסוי:

$$T = 64, U = 55, Z = 32$$

הסימולציות הורצו עבור האלגוריתמים Greedy GC ו-Greedy Lookahead. עבור כל מדידה נלקח פרמטר $N$ בטווח שבין $10^5 - 10^6$. כמות המחיקות $E$ עבור פרמטר $N$ כלשהו נלקחה כממוצע של

מספר הרצות (מס' ההרצות נע בין 5-20 הרצות כתלות ב-$N$. מס' ההרצות ירד ככל ש-$N$ גדל כדי לחסוך בזמן הריצה של הסימולציה). תוצאות הניסוי מוצגות בגרף:




ניתן לראות מתוצאות הניסוי כי אכן $\Delta E$ גדל ככל ש-$N$ גדל. בכך אנחנו מאששים את ההשערה שלנו ויכולים להגיע למסקנה שאלגוריתם Greedy Lookahead מביא לשיפורים לוקליים בכל מחזור GC ובכך מקטין את מספר המחיקות הכולל ומקטין את ה-write amplification.

משימות המשך להרחבת הפרויקט:

1. אופטימיזציה לאלגוריתם Greedy Lookahead – בחירת הבלוק האופטימלי למחיקה מתוך טווח גדול יותר של בלוקים. נרצה לבחון האם ניתן להסתכל לא רק על הבלוקים בקבוצה $V[Y]$, אלא גם על הבלוקים בקבוצה $V[Y+1]$ וכו'.. בנוסף נבצע התאמות בטווח החיפוש על מנת לשפר את סיבוכיות חישוב של הפונקציה $block\_score$.
2. אופטימיזציה לאלגוריתם Greedy Lookahead – בחירת הפונקציה $block\_score$. נרצה לבחון וריאציות שונות של הפונקציה $block\_score$ ולבחון את השפעתן על ה- write amplification המתקבל.
3. הצגת ניסויים המשווים בין האופטימיזציות בסעיפים 1 ו-2 עם המימוש המקורי של Greedy Lookahead ועם אלגוריתם Greedy GC המקורי.
4. בחינת ביצועי האלגוריתם כאשר התפלגות הכתיבות היא Hot/Cold.

## Writing Assignment GC

רעיון כללי ומוטיבציה – נרצה כעת לבחון גישה שונה עבור כתיבה של דפים לזיכרון. בגישה זו, נבצע את הכתיבות במקטעים, ונגדיר את המושג של "חלון" כתיבות אשר יגדיר את הגודל של כל מקטע. נפעיל אלגוריתם אשר יחשב עבור כל דף בחלון הכתיבות מהו הבלוק האופטימלי אליו נרצה לכתוב את הדף. בכך ניצור מראש שיבוץ של כל דף אל הבלוק אליו הוא נכתב, כאשר מחיקת הבלוקים מתבצעת תוך כדי on-demand.

אלגוריתם זה יוצג במלואו כחלק מהרחבת הפרויקט:

1. הצגה מלאה של המוטיבציה לאלגוריתם והרעיון הכללי
2. תיאור פורמלי של האלגוריתם
3. כוונון פרמטרים
4. ניסויים והשוואה מול אלגוריתם Greedy GC עבור כתיבות יוניפורמיות
5. ניסויים והשוואה מול אלגוריתם Greedy GC עבור התפלגות כתיבות Hot/Cold
6. מימוש האלגוריתם בסימולטור

## Generational GC

רעיון כללי ומוטיבציה

נרצה לצלול כעת עמוק יותר אל תוך ההגדרה של "תוחלת חיים" של דף, ולאפיין התנהגות של דפים לפי תוחלת החיים שלהם. ליתר דיוק, תוחלת החיים תוגדר עבור **כתיבה** של דף ולא עבור הדף עצמו. דהיינו, לדף לוגי יכול להיות גיל שונה בכל פעם שהוא נכתב כחלק מרצף הכתיבות. היתרון המשמעותי ביותר שיש לנו תחת המודל הנוכחי בהשוואה למודל הסטנדרטי, הוא שיש לנו יכולת לדעת עבור כל דף שנכתב, מתי הפעם הבאה שהוא הולך להיכתב. מכאן נוכל להגדיר באופן פורמלי את תוחלת החיים של כתיבה כלשהי:

הגדרה 1:

תוחלת החיים של הכתיבה ה-$i$ ב-$ws$ תסומן ב-$age(i)$, ותוגדר באופן הבא:

$$next(i) = \begin{cases} j, & \exists j \; s.t \; ws[i] = ws[j] \; and \; i < j \\ N, & else \end{cases}$$
$$age(i) = next(i) - i$$

למעשה, תוחלת החיים של הכתיבה שווה למס' הכתיבות שיעברו עד הפעם הבאה שאותו הדף ייכתב, כלומר עד הרגע בו הכתיבה הנוכחית תידרס. במקרה בו הכתיבה לא תידרס, תוחלת החיים שלה תוגדר להיות המרחק מ-$N$ (אנחנו מניחים שלאחר הכתיבה האחרונה הזיכרון כולו נמחק).

טרמינולוגיה – נשתמש במושג דף "חי" כדי להתייחס לדף שנמצא במצב valid, ובמושג דף "מת" כדי להתייחס לדף הנמצא במצב invalid.

דוגמה: נסתכל על רצף כתיבות לדוגמה עבור $N = 20, U = 10$:

| **write no.** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lpn | 1 | 9 | 0 | 2 | 3 | 5 | 7 | 5 | 4 | 0 | 2 | 3 | 6 | 1 | 7 | 8 | 4 | 5 | 2 | 3 |

נחשב מס' ערכי $age$ לדוגמה:

| $write\ no.$ | $lpn$ | $age$ |
|---|---|---|
| 0 | 1 | 13 |
| 1 | 9 | 18 |
| 2 | 0 | 7 |
| 3 | 2 | 7 |
| 4 | 3 | 7 |
| 5 | 5 | 2 |
| 10 | 2 | 8 |
| 11 | 3 | 8 |

כעת, נסתכל על רצף הכתיבות הנתון לעיל. נשים לב כי דף מס' 2 ודף מס' 3 תמיד נכתבים אחד אחרי השני וגילם זהה. מכאן אנחנו יכולים להסיק כי הדפים הללו חיים ביחד וגם "מתים" ביחד. לכן, נרצה לדאוג שהדפים הללו ייכתבו לאותו הבלוק. בכך, כל עוד הדפים חיים הם ימלאו בלוק ויתרמו לנצילות של הזיכרון. ברגע שהדפים ימותו, הם ימותו בסמוך אחד לשני ובכך יתרמו להקטנת מספר הדפים הולידיים בבלוק, ובכך ימזערו את ה-penalty שיש לשלם על מחיקת הבלוק. אם נכליל את ההתנהגות של שני הדפים הנ"ל לבלוק שלם – נוכל לקבל בלוק בו כל הדפים הם "בני אותו דור", **וישמרו את ההתנהגות לפיה הם חיים ומתים יחד.**

למעשה, ניתן לנסח כאן שתי אינווריאנטות משלימות אותן ננסה לשמר:

1. דף שנמצא בתחילת חייו ישאף להיכתב לבלוק עם כמה שיותר דפים חיים.
2. דף שנמצא בסוף חייו ישאף להיות בבלוק עם כמה שיותר דפים מתים.

ננסה להסביר פורמלית את המוטיבציה להגדרות אלו. אינווריאנטה מס' 2 היא טריוויאלית – דף שנמצא בסוף חייו עתיד להפוך להיות במצב invalid, ולכן ככל שיהיו יותר דפים מתים באותו הבלוק, כך הבלוק יהפוך להיות מועמד טוב יותר למחיקה – ערך ה-$Y$ קטן וממסקנה 1 נקבל כי ה- write amplification יקטן גם כן.

על מנת להראות את המוטיבציה מאחורי אינווריאנטה מס' 1, נסתכל על memory layout כללי בו הזיכרון נמצא במצב steady state, וכל הבלוקים מלאים כאשר 80% מהתפוסה הינה של דפים חיים (valid) ו-20% מהתפוסה היא של דפים מתים (invalid). בה"כ נניח כי הפיזור של הדפים המתים מתחלק שווה בשווה בין כל הבלוקים (זהו למעשה המצב הגרוע ביותר). המחשה של תמונת הזיכרון:

במצב זה $Y = 8$. כעת נניח כי אחוז התפוסה של הזיכרון נשאר זהה, אך בלוק 0 מכיל רק דפים חיים. במצב זה 20% מהדפים המתים יתפזרו בין שאר הבלוקים ונקבל:

| $Blcok\ 0$ | $Blcok\ 1$ | $Blcok\ 3$ | $Blcok\ 4$ |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

כעת $Y = 7$, לכן ממסקנה 1 נקבל כי שמירה על אינווריאנטה 1 מביאה לירידה ב-write amplification.

<u>תיאור האלגוריתם</u>

אלגוריתם זה הוא למעשה אלגוריתם לכתיבה של דפים. לצורך תיאור האלגוריתם, נגדיר שני בלוקים ייעודיים אשר ישמשו לכתיבה של דפים מדור 1 (הדור הצעיר), ודפים מדור 2 (הדור המבוגר). בלוקים אלו יתמלאו בהדרגה, ובכל פעם שאחד מה-generational blocks יתמלא ייבחר בלוק חדש להחליף את מקומו.

Suppose: $ws$ of size $N$, where $ws[i]$ is the i[th] logical page number to be written to memory.

Initial state:

1. Define: $youngGen = oldGen = NIL$

Given a logical page $lpn_i \in [0, U \cdot Z - 1]$:

1. $generation = getGen(i)$
2. $writeToGenerationalBlock(lpn, generation)$

תיאור הפונקציה $getGen(i)$:

1. $pageScore = age(i)$
2. $bound = \frac{Z \cdot U}{2}$
3. $if\ pageScore < bound : return\ youngGen$
4. $return\ oldGen$

תיאור הפונקציה $writeToGenerationalBlock(lpn, generation)$:

1. $genBlock = getGenBlock(generation)$
2. $if\ genBlock = NIL$:
   2.1. $if\ freelist\ is\ empty$:
      2.1.1. call $GC$.
   2.2. $updateGenBlock(generation, freelist.pop())$
3. Write $lpn$ to $genBlock$ and update FTL accordingly.
4. $if\ genBlock\ block\ is\ full$:
   4.1. $updateGenBlock(generation, NIL)$

הפונקציה $updateGenBlock$ מקבלת פרמטר שאומר איזה דור צריך לעדכן (young/old) וערך חדש לשים בתור ה-generation block הרלוונטי.

<u>מימוש האלגוריתם</u>

אלגוריתם Greedy Lookahead ממומש בסימולטור. הוראות ההפעלה נמצאות בקובץ README.md.

הערה – במימוש האלגוריתם בסימולטור, ניתן להכניס כקלט את מספר הדורות עבור הסימולציה (מומש לצורך הרחבת הפרויקט שתוצג בהמשך). על מנת להריץ את הסימולציה המתאימה לתיאור האלגוריתם כפי שמובא לעיל יש לבחור ב-2 דורות.

<u>אופטימזציות והרחבה האלגוריתם</u>

במסגרת דו"ח זה לא נעסוק בהרחבות ואופטימיזציות לאלגוריתם זה, אך בכל זאת נציג שיפור קטן ומשמעותי שנוכל להכניס אל האלגוריתם כבר בשלב זה. כפי שניתן לראות מתיאור האלגוריתם, בכל פעם שאחד מה-generation blocks מתמלא, אנחנו מחפשים בלוק פנוי ה-freelist כדי להגדירו בתור ה-generation block החדש עבור הדור הרלוונטי. במקרה בו ה-freelist ריק, נקרא לאלגוריתם GC על מנת לפנות בלוק מלא. במימוש הנאיבי פעולת ה-GC תיעשה באמצעות אלגוריתם Greedy GC הסטנדרטי. נשים לב כי ניתן להחליף את הקריאה ל-Greedy GC בקריאה ל- Greedy Lookahead GC, אשר יעילותו כבר הוכחה בחלקים הקודמים.

אם כך, הפונקציה $writeToGenerationalBlock(lpn, generation)$ החדשה תיראה כך:

1. $genBlock = getGenBlock(generation)$
2. $if\ genBlock = NIL$:
   2.1. $if\ freelist\ is\ empty$:
      2.1.1. call $\boldsymbol{GCLookAhead}$.
   2.2. $updateGenBlock(generation, freelist.pop())$
3. Write $lpn$ to $genBlock$ and update FTL accordingly.
4. $if\ genBlock\ block\ is\ full$:
   4.1. $updateGenBlock(generation, NIL)$

## תוצאות וניסויים

נרצה לבחון את ביצועי האלגוריתם החדש אל מול אלגוריתם Greedy GC המקורי ומול Greedy Lookahead.

הניסוי הראשון יהיה זהה לניסוי מהחלק הראשון ויבחן את השינוי ב-write amplification כתלות ב-over-provisioning. נזכיר את הפרמטרים לניסוי:

$$T = 64, Z = 32, N = 10^5$$

כל מדידה בניסוי היא של ערך write amplification. המדידה נקבעה על ידי לקיחת ממוצע של 20 הרצאות עבור הפרמטרים הרלוונטיים. שתי העמודות הראשונות הן העמודות מהניסוי שהוצג בחלק הראשון עבור Greedy Lookahead. נוסיף כעת עמודה נוספת עם תוצאות הניסוי עבור אלגוריתם Generational GC:

| ALGORITHM / OP | GREEDY GC WA | GREEDY LOOKAHEAD WA | GENERATIONAL WA |
|---|---|---|---|
| $U = 60, OP = 0.0666$ | 6.78036 | 6.65409 | 6.6538 |
| $U = 56, OP = 0.1428$ | 3.81101 | 3.77215 | 3.73817 |
| $U = 52, OP = 0.2307$ | 2.69367 | 2.67263 | 2.62013 |
| $U = 48, OP = 0.3333$ | 2.11129 | 2.09632 | 1.9509 |
| $U = 44, OP = 0.4545$ | 1.75364 | 1.74381 | 1.54302 |
| $U = 40, OP = 0.6$ | 1.51706 | 1.50927 | 1.31184 |

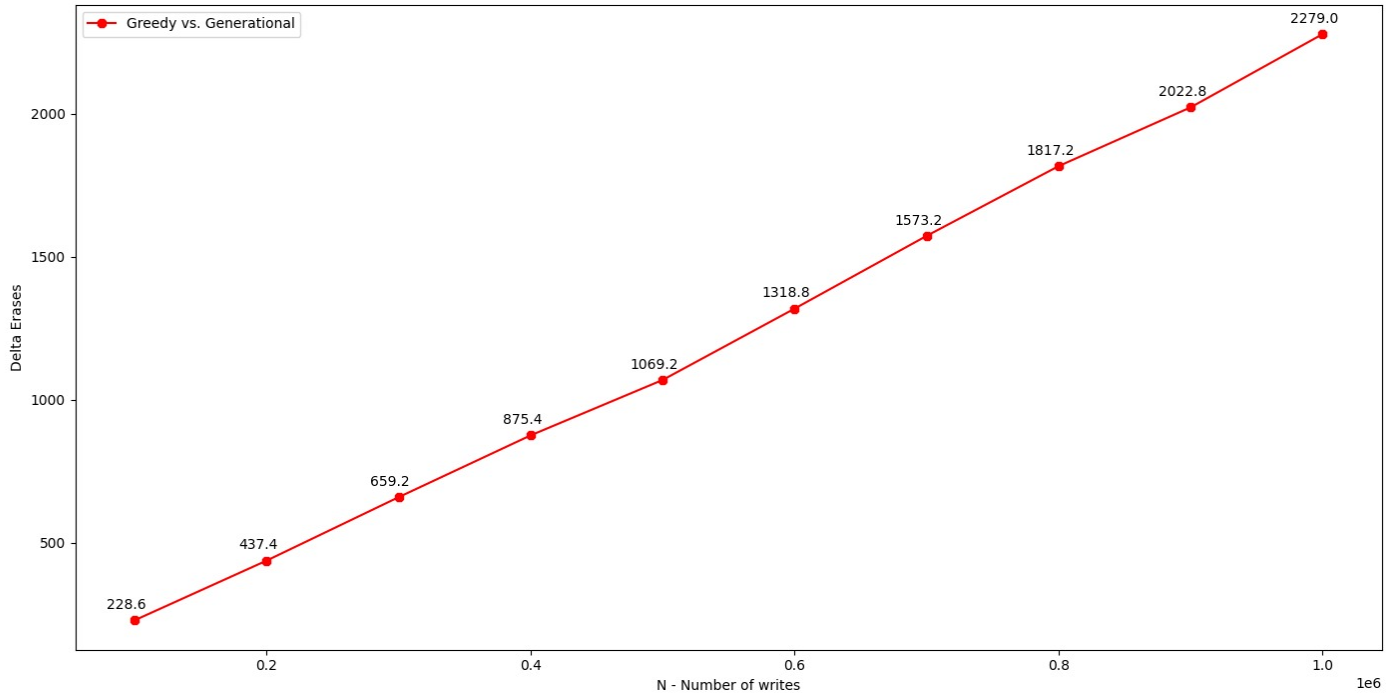
ניתן לראות כי האלגוריתם משיג שיפור בערכי ה-write amplification בצורה עקבית, לכל ערכי ה-OP השונים. ניתן לראות כי השיפור מושג גם בהשוואה לאלגוריתם Greedy GC המקורי וגם בהשוואה ל-Greedy Lookahead. תוצאה זו אינה מפתיעה במיוחד, מכיוון שאלגוריתם ה-Generational מכיל במובן מסוים את Greedy Lookahead, ולכן נצפה שלכל הפחות יהיה טוב כמוהו.

בניסוי השני נמדוד שוב את המדד: $\Delta E = E_{Greedy} - E_{Lookahead}$. כזכור, מדד זה מתאר את ההפרש בכמות המחיקות בין אלגוריתם Greedy GC לבין אלגוריתם Greedy Lookahead. הפרמטרים עבור הניסוי:

$$T = 64, U = 55, Z = 32$$

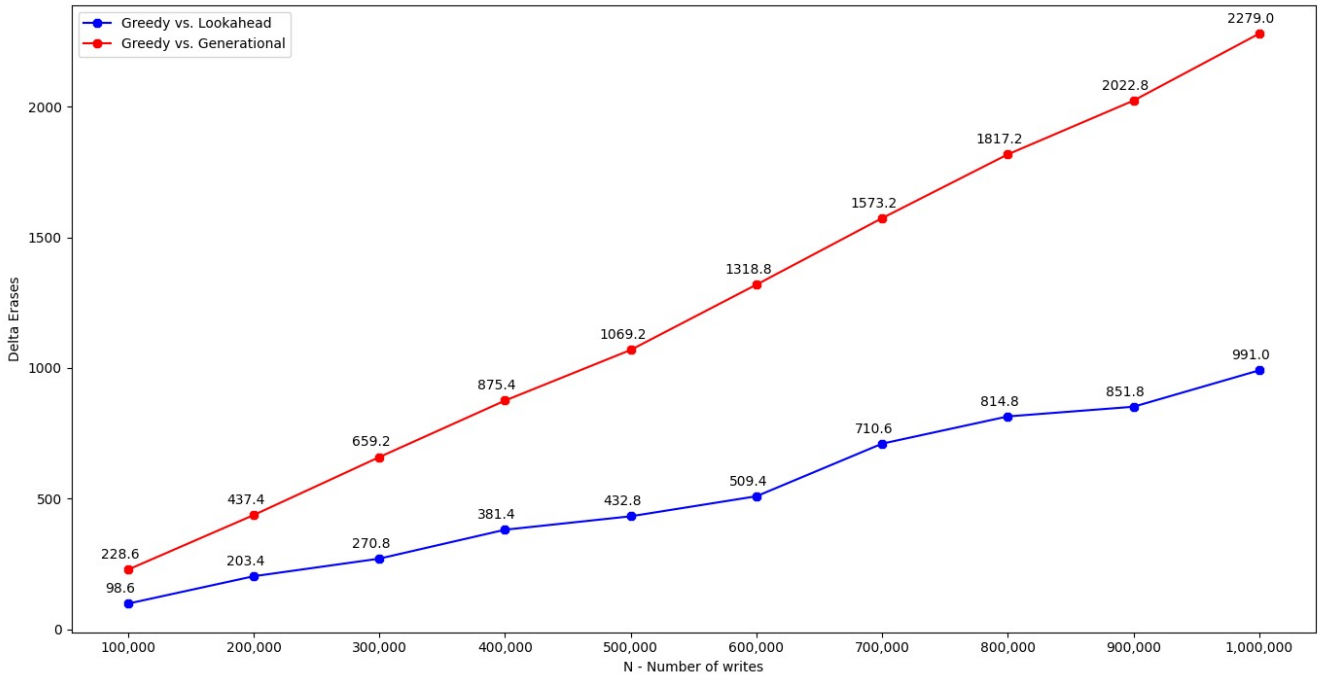
שאר פרטי הניסוי זהים לניסוי שהורץ בחלק הראשון עבור Greedy Lookahead.

תוצאות הניסוי מוצגות בגרף :




ניתן לראות כי בדומה לתוצאות עבור Greedy Lookahead, גם כאן השיפור במספר המחיקות גדל כתלות בכמות הכתיבות $N$.

על מנת להדגים את ההבדל בין Greedy Lookahead לבין Generational GC, נציג את תוצאות שני הניסויים על אותה מערכת צירים :



כפי שניתן לראות, השיפור שמשיג אלגוריתם ה-Generational במספר המחיקות הוא המשמעותי ביותר.

משימות המשך להרחבת הפרויקט:

1. בחירת מספר הדורות עבור אלגוריתם Generational – בחירת מספר הדורות נעשתה בצורה נאיבית למדי בגרסה הנוכחית של האלגוריתם. נרצה להרחיב את האלגוריתם ולאפשר מס' דורות גדול יותר על מנת לקבל גרנולריות יותר טובה עבור חלוקת הכתיבות.
2. בחירת האינטרוול עבור החלוקה של דפים בין הדורות השונים.
3. מימוש אלגוריתם Generational GC עם תמיכה במספר דורות משתנה.
4. ביצוע ניסויים לבחינת ביצועי האלגוריתם כתלות במס' הדורות. נרצה לכוונן את פרמטר מס' הדורות ולבצע ניסויים כדי למצוא את מס' הדורות האופטימלי כתלות בפרמטרים השונים של הזיכרון.
5. השוואה אל מול האלגוריתמים הקודמים (Greedy GC, Greedy Lookahead, Writing Assignment).
6. בחינת ביצועי האלגוריתם כאשר התפלגות הכתיבות היא Hot/Cold.

## <u>סיכום ומסקנות</u>

בפרויקט זה ניסינו להראות כיצד ניתן להשיג שיפור במדד ה-write amplification עבור זכרונות SSD. המוטיבציה לבעיה הינה הקטנת מספר המחיקות של בלוקים, פרמטר אשר משפיע באופן ישיר על חיי הזיכרון ועל ביצועיו. ניסינו לתקוף את הבעיה ממספר כיוונים, ונוכחנו לגלות כי גם בהינתן הידע על כל הכתיבות מראש, המשימה לתכנון ולממש אלגוריתם אשר מביא לשיפור בפרמטרים החשובים לנו לא הייתה טריוויאלית כלל. על פניו, המודל והאלגוריתמים שהוצגו כאן הינם תאורטיים בלבד, משום שידע מקדים על כתיבות הוא מודל לא פרקטי עבור העולם האמיתי. אך ניתן לחשוב על מקומות בהם הרעיונות והאלגוריתמים שהצגנו כן יכולים להיות שימושיים. לדוגמה, במערכות רבות הכתיבות לדיסק לא מבוצעות בבת אחת אלא נכתבות ל-buffer בזיכרון, ורק לאחר שה-buffer מתמלא מבוצע flush לדיסק. אם כך, נוכל להתייחס ל-buffer בתור ה-$writing\_sequence$ שלנו, ולנצל את העובדה שהמידע נמצא בזיכרון כדי להריץ אלגוריתמי אופטימיזציה ותכנון כתיבה המבוססים על רעיונות שהוצגו כאן. כמובן שגישה זו דורשת טיפול בעוד לא מעט נושאים שמעט הזנחנו כאן, בראש ובראשונה בעיות סיבוכיות הזמן של האלגוריתמים. עם זאת, אנחנו מאמינים כי ניתן להציע רלקסציות נוספות לתנאי המודל על מנת לאפשר זמני כתיבה סבירים.

כחלק מהרחבת הפרויקט ייתכן ונבחן הכללה של המודל שהוצג כאן, בו במקום לקבל את כל הכתיבות מראש בתור $writing\_sequence$, יהיה נתון לנו רק חלון כתיבות בגודל $n < N$. מטרתנו תהיה להתאים את האלגוריתמים השונים לעבודה עם חלון כתיבות במקום עם רצף כתיבות שלם שידוע מראש. נשאף להראות כי ככל ש-$n \to N$ האלגוריתמים מתכנסים לביצועי האלגוריתמים המקוריים שהוצגו עד כה, ואם $n \to 0$ האלגוריתמים מתכנסים לביצועי Greedy GC הסטנדרטי. ייתכן כמובן שהבחירה באיזה אלגוריתם להשתמש תהיה תלויה בגודל החלון הנתון לנו ובכמות הידע על ה-"עתיד" שנוכל לקבל מהחלון.