Garbage Collection Algorithms for Flash Memories

מגישים:

- אייל לוטן, ת"ז: 302288527. מייל: eyallotan@campus.technion.ac.il
- דור סורה, ת"ז: 204098784 מייל: dorsura@campus.technion.ac.il

מנחה: גב' דניאלה בר-לב

מרצה אחראי: פרופ' איתן יעקובי

תוכן עניינים

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

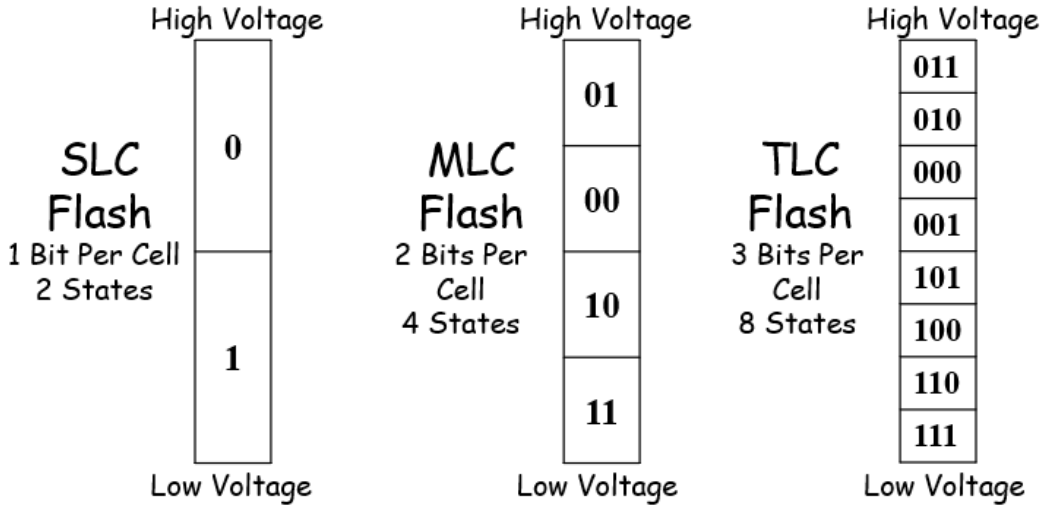
נציג תחילה את המודל התיאורטי בו נעסוק, וכמו כן נציין מספר הגדרות אשר ישמשו אותנו בפרקים הבאים.

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 הזיכרון

זכרונות פלאש שומרים מידע במערך של תאי זיכרון 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 זיכרון.

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.

<u>דוגמה מס' 1: Memory layout של זיכרון פלאש:</u>

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

<u>Write Amplification (WA)</u> – 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.

<u>Erasure Factor (EF)</u> – 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

לזיכרונות 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. נציע אלגוריתמים שונים אשר יעבדו תחת מודל בעל הנחות מקלות ביחס למודל הרגיל (יפורט בהמשך). תחת מודל זה נראה כיצד ניתן להשיג שיפורים ביחס לאלגוריתם אשר נחשב כאופטימלי עבור מודל ההנחות הרגיל.

Greedy GC

תחילה נציג בקצרה את האלגוריתם Greedy GC. **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 $0 \le i \le Z$ 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 האלגוריתם

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.

הצגת המודל וההנחות עבור הפרויקט

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 (עבור Greedy GC הדפים נכתבים לפי האלגוריתם שפורט לעיל).

Steady State Assumption

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 בדף ה-Github של הפרויקט).

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.

ניסויים

בפרקים הבאים יוצגו ניסויים רבים שביצענו על מנת לבחון את ביצועי האלגוריתמים השונים, לאשש או להפריך השערות ולכוונן פרמטרים שונים. כל הניסויים בוצעו תוך הרצת האלגוריתמים על הסימולטור אותו בנינו. מבדיקה של פרקטיקות מקובלות בתעשייה, גילינו שבכל הקשור לביצועי זיכרון, ישנם מקרים רבים בהם ישנו קושי לפתח מודלים מתמטיים אשר מתאימים בדיוק לדרישות הבעיה, ולכן מסקנות מתקבלות כתוצאה מהרצה של ניסויים אמפיריים. בשל כך שמנו דגש נרחב על הרצת ניסויים בפרויקט זה. לפני כל ניסוי יופיע תיאור קצר של מטרת הניסוי, וכן יוצגו התנאים בהם הורץ הניסוי כגון מס' הכתיבות שבוצעו, גודל הזיכרון (מס' בלוקים לוגיים ופיזיים ומס' הדפים בכל בלוק) והגדרות נוספות אשר רלוונטיות לכל ניסוי.

הערה טכנית לגבי הצגת הגרפים – את חלק מתוצאות הניסויים נרצה להציג בצורה גרפית על מנת להדגים מגמות והבדלים בין האלגוריתמים השונים. בגלל שההבדלים המספריים לפעמים מאד קטנים ביחס לרזולוציות של הגרף, חלק מהגרפים יציגו מקטע מצומצם של הניסוי בו ניתן להבחין בהבדלים הרלוונטיים לאותו הניסוי בצורה הטובה ביותר.

מכיוון מטרת הפרויקט היא לבצע השוואות על מול אלגוריתם Greedy GC, נרצה ליצור בסיס השוואתי קבוע אליו נוכל להשוות את האלגוריתמים השונים בהם נעסוק.

ניסוי מס' 1 – אלגוריתם Greedy GC

- תיאור הניסוי – הרצת אלגוריתם Greedy GC על מנת לבסס בסיס השוואתי עבור אלגוריתמים שונים. נריץ את האלגוריתם עבור ערכי OP שונים ונתעד את ערכי ה-WA ומספר המחיקות.
- תנאי הניסוי – $N = 10^5, T = 64, Z = 32$. כל ערך תוצאה נלקח כממוצע של 100 הרצות.
- תוצאות הניסוי – נציג את התוצאות בטבלה:

OP	0.066	0.142	0.230	0.333	0.454	0.6	0.777	1	1.285	1.666	2.2	3	4.333
WA	6.781	3.811	2.694	2.109	1.754	1.517	1.351	1.231	1.143	1.079	1.035	1.007	1
Erases	21190	11910	8419	6591	5480	4741	4221	3847	3571	3372	3233	3145	3125

נעבור כעת להציג 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?

<u>דוגמה 2 – Greedy Lookahead</u>:

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

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

כשנגיע לכתוב את דף מס' 9, נשים לב כי מתקיים $Y = 0$ ו-$V[Y] = \{1\}$. כלומר, אנחנו לא משלמים קנס 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 נדרש 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.

תיאור האלגוריתם

ניגש כעת לתיאור האלגוריתם Greedy Lookahead:

Suppose: ws of size N, where $ws[i]$ is the ith 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: $blockScore(B, ws, i)$:

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

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 בדף ה-Github של הפרויקט.

תוצאות וניסויים

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

ניסוי מס' 2 – Greedy Lookahead

- תיאור הניסוי – נריץ את אלגוריתם Greedy Lookahead ונבחן את השינוי ב- write amplification כתלות ב-over-provisioning.

- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. כל מדידה בניסוי היא של ערך write amplification עבור ערך OP מסוים. המדידה נקבעה על ידי לקיחת ממוצע של 20 הרצאות עבור הפרמטרים הרלוונטיים.
- תוצאות הניסוי – נציג את תוצאות ה-WA בטבלה עם השוואה לאלגוריתם Greedy GC הקלאסי:

ALGORITHM / OP	GREEDY GC	GREEDY LOOKAHEAD
0.0666	6.78079	6.70131
0.1428	3.81117	3.78392
0.2307	2.69403	2.68005
0.3333	2.1092	2.10211
0.4545	1.75361	1.7487
0.6	1.51698	1.51163
0.777	1.35085	1.34639
1	1.2309	1.22644
1.285	1.14286	1.13786
1.666	1.07919	1.07342
2.2	1.03464	1.02795
3	1.00655	1.00345
4.333	1	1

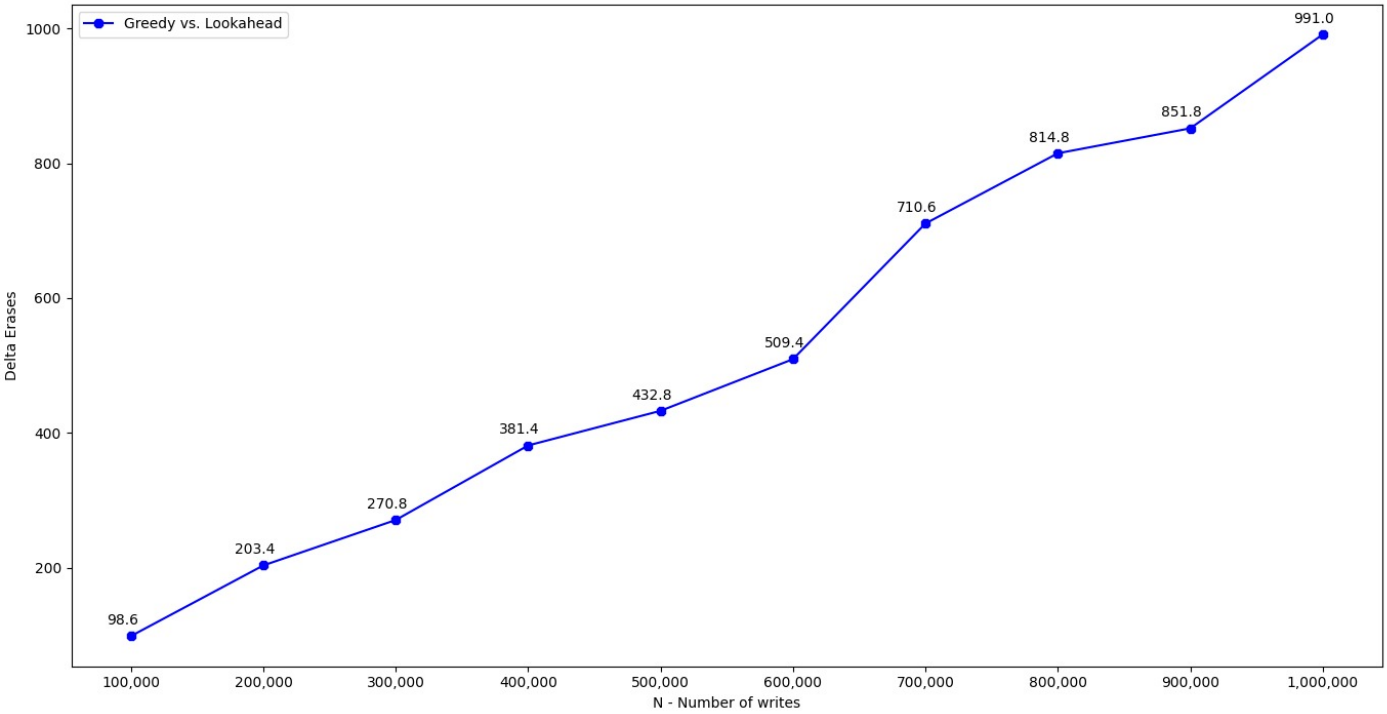
ניתן לראות כי אכן האלגוריתם החדש משיג שיפור בערכי ה-write amplification בצורה עקבית, לכל ערכי ה-OP השונים. תוצאה זו אינה מפתיעה – מטרתנו באופטימיזציית ה-Lookahead היא לבצע שיפורים **לוקליים**, אשר יבטיחו כי בכל הפעלה של ה-GC נבחר בלוק למחיקה שיהיה לכל הפחות טוב כמו הבחירה שהיה בוחר אלגוריתם Greedy GC 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.

ניסוי מס' 3 – מדידת ΔE

- תיאור הניסוי – מדידת השינוי ב-ΔE כתלות במספר הכתיבות N.
- תנאי 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 – נציג את תוצאות הניסוי בגרף:



ניתן לראות מתוצאות הניסוי כי אכן ΔE 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.

הרחבות ואופטימיזציות לאלגוריתם Greedy Lookahead

עד כה הצגנו את המימוש הנאיבי של אלגוריתם ה-Greedy Lookahead. נרצה כעת להרחיב את האלגוריתם ולבצע ניסויים לכוונון פרמטרים שונים. בנוסף, נתעניין בדרכים בהן נוכל לשפר את סיבוכיות הזמן של האלגוריתם מבלי לפגוע בצורה משמעותית בביצועים.

פונקציית Block score

תזכורת: בתיאור האלגוריתם Greedy Lookahead נעשה שימוש בפונקציית Block score (בקיצור – BS). כזכור, בהינתן בלוק B, הניקוד שיינתן לבלוק מבטא את "תוחלת החיים" של הדפים בו. ככל שהציון של הבלוק גבוה יותר, כך תוחלת החיים של הדפים בו ארוכה יותר ולכן נרצה לבחור אותו למחיקה בתהליך ה-GC.

Define: $blockScore(B, ws, i)$:

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

בהגדרת הפונקציה BS, ניתן לראות כי הציון הניתן לבלוק תלוי במספר הדפים הולידיים אשר נמצאים בבלוק בכל איטרציה. למרות שפונקציה זו אכן משיגה את המטרה שהוגדרה לעיל, ישנן מספר נק' בעייתיות אשר נשאף לשפר בחלק זה. נציג תחילה את הבעיות איתן ננסה להתמודד:

- התאמת פונקציית ה-BS כך שתתחשב בקצב דעיכת הבלוק – במימוש המקורי של פונקציית ה-BS, הציון שניתן לבלוק הינו פונקציה של מספר הדפים הולידיים אשר נותרו בבלוק בכל מחזור כתיבה עתידי החל מהכתיבה הנוכחית בה אנחנו נמצאים. פונקציה זו אינה מתחשבת בזמן ובקצב הדעיכה של הבלוק, ובכך היכולת שלה לזהות נכונה את הבלוק המתאים למחיקה נפגעת. כאשר אנחנו מדברים על קצב הדעיכה של הבלוק, הכוונה היא למהירות בה מס' הדפים הולידיים אשר נותרו בבלוק שואף לאפס.

  <u>דוגמה מס' 3 – פונקציית BS</u>: נניח כי הפעלנו את פונקציית ה-BS בגרסתה המקורית (כפי שמוצגת לעיל) על שני בלוקים B_1, B_2. מכיוון ששני הבלוקים האלו נמצאים בקבוצה $V[Y]$,

אזי יש להם אותו מספר של דפים ולידיים בנק' הזמן בה מבוצע החישוב. נניח בה"כ שהדפים הולידיים בכל אחד מהבלוקים הינם: $Valid_{B_2} = \{6,7,8,9,10\}$, $Valid_{B_1} = \{1,2,3,4,5\}$.
כעת, נסתכל על חישוב ה-BS עבור כל אחד מהבלוקים הנ"ל. כזכור, נמשיך לסכום את התוספות ל-BS עבור כל אחד מהבלוקים כל עוד יש בו דפים ולידיים (או שהגענו לסוף ה-ws).
כעת, נניח שהתנהגות הבלוק B_1 עם הזמן היא כזו (כל חץ מייצג איטרציה עתידית של כתיבה):

$$Valid_{B_1} = \{1,2,3,4,5\} \to \{1,2,3,4,5\} \to \{1,2,3,4,5\} \to \{1,2,3,4,5\} \to \{1,2,3,4,5\}$$
$$\to \{1,2,3,4\} \to \{1,2,3\} \to \{1,2\} \to \{1\} \to \emptyset$$

חישוב ה-BS של B_1 יניב תוצאה של $BS(B_1) = 35$. כעת נסתכל על התנהגות הבלוק B_2 עם הזמן:

$$Valid_{B_2} = \{6,7,8,9,10\} \to \{6,7,8,9\} \to \{6,7,8\} \to \{6,7\} \to \{6\} \to \{6\} \to \{6\}$$
$$\to \underbrace{\cdots}_{18\ iterations} \to \{6\} \to \emptyset$$

חישוב ה-BS של B_2 יניב תוצאה של $BS(B_2) = 36$. לכן, בהינתן שאלו שני הבלוקים היחידים ב-$V[Y]$, בלוק B_2 הוא זה שייבחר למחיקה. בחירה זו היא כמובן לא הבחירה האידיאלית בסיטואציה זו, מכיוון שאנחנו בוחרים למחוק בלוק ולהעתיק דפים אשר גם ככה צפויים להימחק באיטרציות הקרובות, ובכך אנחנו פוגעים ב-WA (כפי שהראינו בדוגמה מס' 2). ננסה למדל את ההתנהגות הבעייתית של בלוק B_2 אשר גרמה לנו לבחור בו – קצב הדעיכה של הבלוק היה מאד מהיר, אך עדיין נשארו בו מעט דפים ולידיים אשר נותרו חיים למשך הרבה איטרציות והמשיכו להוסיף משקל לציון הכולל של הבלוק. מנגד, B_1 מציג קצב דעיכה איטי יותר אך לבסוף מתרוקן לחלוטין בשלב מוקדם יותר מבלוק B_2 ולכן מפסיק לצבור תרומות לציון שלו. בנוסף אנחנו שמים לב כי התרומה של הדף הבודד שנשאר חי לציון הבלוק לא משתנה עם הזמן. מכיוון שההחלטה למחיקת הבלוק צריכה להתקבל בנקודת הזמן בהווה בה אנחנו נמצאים, פחות יעניינו אותנו תרומות לציון הבלוק אשר מגיעות מהעתיד "הרחוק", ויותר נתעניין דווקא בתרומות אשר מגיעות מהעתיד "הקרוב" יותר, אשר מושפעות ממצב הזכרון כפי שהוא נראה כעת. מסקנה המתבקשת היא שנרצה להוסיף לפונקציית ה-BS אלמנט אשר מתחשב בקצב הדעיכה של הבלוק.

- סיבוכיות הזמן של הפונקציה – במימוש הנוכחי, פונקציית ה-BS מבצעת סריקה אשר סוכמת את התרומות החל מהמקום ה-i המסמל את מספר הכתיבה הנוכחי ועד לכתיבה ה-N. במקרה הגרוע נקבל סיבוכיות זמן של $O(N)$. זוהי כמובן סיבוכיות זמן לא פרקטית כאשר מדובר במערכת אמיתית. בנוסף, כפי שהוסבר בסעיף הקודם, התחשבות בתרומות המגיעות מהעתיד "הרחוק" מידי עלולות לפעמים אף לפגוע בחישוב ה-BS ולתת משקל גבוה יותר לבלוקים אשר

לא נרצה בהכרח למחוק במחזור ה-GC הנוכחי. מכך אנחנו מגיעים למסקנה כי נרצה להגדיר טווח חישוב קטן יותר עבור פונקציית ה-BS. נשאף לעשות זאת כמובן תוך פגיעה מינימלית (אם בכלל) בביצועים.

- התאמת פונקציית ה-BS ל-OP – המימוש הנוכחי של פונקציית ה-BS אינו מתייחס למדד ה-OP של הזיכרון. נשאף לתקן זאת על ידי שילוב של פונקציונליות בסימולטור ה-GC אשר תדע לבחור את פונקציית ה-BS האידיאלית כתלות ב-OP אשר בוחר המשתמש.

נתחיל בביצוע התאמה של פונקציית ה-BS כך שתתחשב בקצב דעיכת הבלוק – נגדיר מחדש את פונקציית ה-BS באופן הבא:

Define: $blockScore(B, ws, i)$:

1. Define $blockScore = 0$.
2. Define $pagesInBlock = \{p \in [0, U \cdot Z - 1] \mid p \in B \text{ and } p \text{ is valid}\}$.
3. $for(long\ long\ pos = i; pos < N; pos++)$:
   3.1. if $ws[pos] \in pagesInBlock$:
      3.1.1. $pagesInBlock \leftarrow pagesInBlock \setminus \{ws[pos]\}$.
      3.1.2. if $|pagesInBlock| = 0$ – return $blockScore$.
   3.2. $\boldsymbol{blockScore} += \frac{|\boldsymbol{pagesInBlock}|}{(\boldsymbol{pos}-\boldsymbol{i}+1)^{\alpha}}$.
4. Return $blockScore$.

הסבר – נעדכן את התוספת לציון הבלוק בכל איטרציה ע"י חלוקה של מספר הדפים החיים במיקום היחסי של הכתיבה ביחס לאינדקס הבסיס i. הפרמטר α הוא פרמטר אותו נרצה לכוונן בעזרת ניסוי. זהו פרמטר אשר יקבע עוצמת הדעיכה בתוספות אשר יינתנו ל-BS בכל איטרציה. עבור $\alpha = 0$ נקבל את פונקציית ה-BS המקורית כפי שהוגדרה בתחילת פרק זה.

ניסוי מס' 4 – קביעת הפרמטר α

- תיאור הניסוי – נבצע הרצה של אלגוריתם Greedy Lookahead עם פונקציית BS המעודכנת. נבחן ערכי α שונים בטווח $[0,10]$. לכל ערך של α, נבצע הרצה של הסימולטור עבור ערכי OP שונים ונבדוק מהו ה-WA המתקבל.
- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. כל ערך של WA נלקח כממוצע של 10 הרצות.
- תוצאות הניסוי – נציג את תוצאות הניסוי בטבלה:

OP / Alpha	0.066	0.142	0.230	0.333	0.454	0.6	0.777	1	1.285	1.666	2.2	3	4.33
0	6.701	3.784	2.68	2.102	1.749	1.512	1.346	1.226	1.138	1.073	1.028	1.003	1
1	6.498	3.721	2.645	2.079	1.734	1.501	1.338	1.22	1.133	1.07	1.027	1.003	1
2	6.248	3.639	2.611	2.062	1.721	1.494	1.334	1.217	1.131	1.069	1.027	1.003	1
3	6.211	3.63	2.605	2.057	1.718	1.493	1.333	1.216	1.131	1.069	1.027	1.004	1
4	6.21	3.628	2.604	2.057	1.719	1.492	1.333	1.216	1.131	1.069	1.027	1.003	1
5	6.207	3.626	2.604	2.058	1.721	1.493	1.333	1.217	1.131	1.07	1.027	1.003	1
6	6.201	3.625	2.605	2.058	1.72	1.493	1.333	1.216	1.131	1.069	1.027	1.004	1
7	6.2	3.628	2.605	2.058	1.721	1.493	1.333	1.217	1.133	1.071	1.03	1.005	1
8	6.219	3.635	2.614	2.062	1.724	1.495	1.336	1.219	1.135	1.073	1.031	1.005	1
9	6.325	3.663	2.663	2.07	1.728	1.502	1.34	1.223	1.137	1.075	1.032	1.006	1
10	6.397	3.685	2.636	2.078	1.734	1.504	1.342	1.225	1.138	1.076	1.033	1.006	1
Best Alpha	**7**	**6**	**5**	**3**	**3**	**4**	**6**	**4**	**5**	**6**	**4**	**5**	**/**

ניתן לראות מתוצאות הניסוי כי פונקציית ה-BS המעודכנת אכן מצליחה להביא לשיפור במדד ה-WA. ניתן לראות כי עבור כל ערכי ה-OP, הפרמטר α הנבחר תמיד גדול מ-0, כלומר תמיד נקבל שעדיף להשתמש בפונקציית ה-BS החדשה עם פרמטר $\alpha > 0$ מאשר פונקציית BS המקורית ($\alpha = 0$).

ניתן לראות מתוצאות הניסוי כי עבור ערכי OP שונים מתקבל פרמטר α אופטימלי שונה. נרצה להשתמש בתוצאה זו ולהרחיב את הסימולטור שלנו כך שיידע לתת למשתמש את הפרמטרים הטובים ביותר עבור כל הרצה. לשם כך הרחבנו את קוד הסימולטור ושילבנו בתוכו בחירה דינמית של פרמטר α כתלות בערך ה-OP איתו הסימולטור מאותחל. בכל פעם שהמשתמש מזין את הפרמטרים עבור הרצת סימולציה כלשהי, נחשב את ערך ה-OP ונאתר את ערך הקרוב ביותר מבין הערכים הרשומים בטבלה לעיל. לאחר שמצאנו את ערך המטרה, נבחר את הפרמטר α האופטימלי (Best Alpha) והוא זה שישמש לחישובי ה-BS בעת הרצת הסימולציה.

כעת, נוכל להוציא מתוך הטבלה לעיל את התוצאות האופטימליות עבור שימוש בפונקציית ה-BS החדשה. נסכם את התוצאות בטבלה:

ALGORITHM / OP	GREEDY GC	GREEDY LOOKAHEAD	GREEDY LOOKAHEAD (NEW BS FUNCTION)
0.0666	6.78079	6.70131	6.20362
0.1428	3.81117	3.78392	3.62675
0.2307	2.69403	2.68005	2.60431
0.3333	2.1092	2.10211	2.05763
0.4545	1.75361	1.7487	1.71999
0.6	1.51698	1.51163	1.49258
0.777	1.35085	1.34639	1.33301
1	1.2309	1.22644	1.2169
1.285	1.14286	1.13786	1.13152
1.666	1.07919	1.07342	1.06942
2.2	1.03464	1.02795	1.02719
3	1.00655	1.00345	1.00342
4.333	1	1	1

ניתן לראות שהצלחנו להביא לשיפור משמעותי בערכי ה-WA הממוצעים ביחס למימוש הראשוני של אלגוריתם Greedy Lookahead.

כעת נרצה להתמודד עם סוגיית סיבוכיות זמן הריצה של הפונקציה לחישוב Block score. נשים לב כי בכל פעם שאנחנו מבצעים חישוב של BS עבור בלוק מסוים, אנחנו סורקים החל מהאינדקס i המסמל את מספר הכתיבה הנוכחית (נקרא לאינדקס זה אינדקס הבסיס) ועד לאינדקס N המסמל את הכתיבה האחרונה ברצף הכתיבות. אמנם קיים תנאי עצירה באלגוריתם אשר יסיים את החישוב במידה ולא נותרו עוד דפים חיים בבלוק אותו אנחנו כרגע בודקים, אך במקרה הכללי הסריקה עד האינדקס N מאריכה את סיבוכיות זמן הריצה בצורה משמעותית – עד ל-$O(N)$ במקרה הגרוע. בנוסף, ראינו כי סריקה של כתיבות הנמצאות בעתיד "הרחוק" לא בהכרח תורמת לדיוק האלגוריתם, ויכולה אף לעיתים להביא לתופעות שיביאו לבחירת הבלוק **הלא רצוי למחיקה**. העדכון שביצענו לפונקציית ה-BS גם הוא מרמז לנו כי הסריקה של כתיבות בעתיד הרחוק עשויה להיות מיותרת – התרומה לציון הבלוק עבור דפים חיים שנמצאים בעתיד הרחוק הולכת ושואפת ל-0 ככל ש-N גדל, ולכן נרצה להגביל את הסריקה שלנו לטווח אשר יקיים את שני התנאים הבאים:

1. סיבוכיות זמן הריצה הכוללת תקטן בצורה משמעותית.

2. התרומות לכל דף חי בטווח לא יהיו זניחות.

נציין שוב כי פונקציית ה-BS במימוש הנוכחי שלה לרוב לא תסרוק עד לאינדקס N, וזאת משום שקיים לנו תנאי עצירה אשר מסיים את הסריקה ברגע שנגיע לכתיבה עתידית בה כל הדפים אשר היו חיים בעת הכתיבה ה-i כבר אינם חיים. עם זאת, עדיין יכולים להיות מקרים בהם טווח הסריקה כן ישאף ל-N . דוגמה פשוטה למקרה כזה היא עבור התפלגות כתיבות Hot/Cold. במקרה זה, דף אשר מוגדר כדף "קר" עלול להישאר חי בבלוק לכל אורך רצף הכתיבות, וכך הסריקה של הבלוק בו נמצא אותו דף קר תיקח זמן רב. מכיוון שאנחנו מניחים כי רוב הדפים הם דפים קרים, נקבל כי ישנו סיכוי גדול כי נאלץ לסרוק עד האינדקס N עבור כל בלוק עליו תופעל הפונקציה BS.

נציע כעת את השינוי לטווח החיפוש:

Define: $blockScore(B, ws, i)$:

1. Define $blockScore = 0$.
2. Define $pagesInBlock = \{p \in [0, U \cdot Z - 1] \mid p \in B \text{ and } p \text{ is valid}\}$.
3. $\boldsymbol{for(long\ long\ pos = i; pos < i + T \cdot Z \text{ and } pos < N;\ pos++)}$:
   3.1. if $ws[pos] \in pagesInBlock$:
      3.1.1. $pagesInBlock \leftarrow pagesInBlock \setminus \{ws[pos]\}$.
      3.1.2. if $|pagesInBlock| = 0$ – return $blockScore$.
   3.2. $blockScore += \frac{|pagesInBlock|}{(pos-i+1)^{\alpha}}$.
4. Return $blockScore$.

הסבר – נצמצם את הטווח הסריקה שלנו ל-$T \cdot Z$ הכתיבות העתידיות. האינטואיציה מאחורי הבחירה הזו בכך שאנחנו לוקחים טווח כתיבות אשר מייצג "מחזור שלם" של כתיבות לזיכרון. בכך אנחנו מצליחים לבסס את ההחלטה שלנו על סמך כתיבות עתידיות אשר בסבירות גבוהה ישפיעו על מבנה הזיכרון **כפי שהוא נראה ברגע הכתיבה ה-i** . בנוסף, בחירה זו מאפשרת לנו לחסום את סיבוכיות הזמן במקרה הגרוע ע"י $O(T \cdot Z)$. תחת ההנחה הסבירה כי במקרים רבים $T \cdot Z \ll N$, זהו שיפור דרמטי בסיבוכיות.

נרצה לאשש כי קיצור הטווח אינו פוגע בביצועי האלגוריתם.

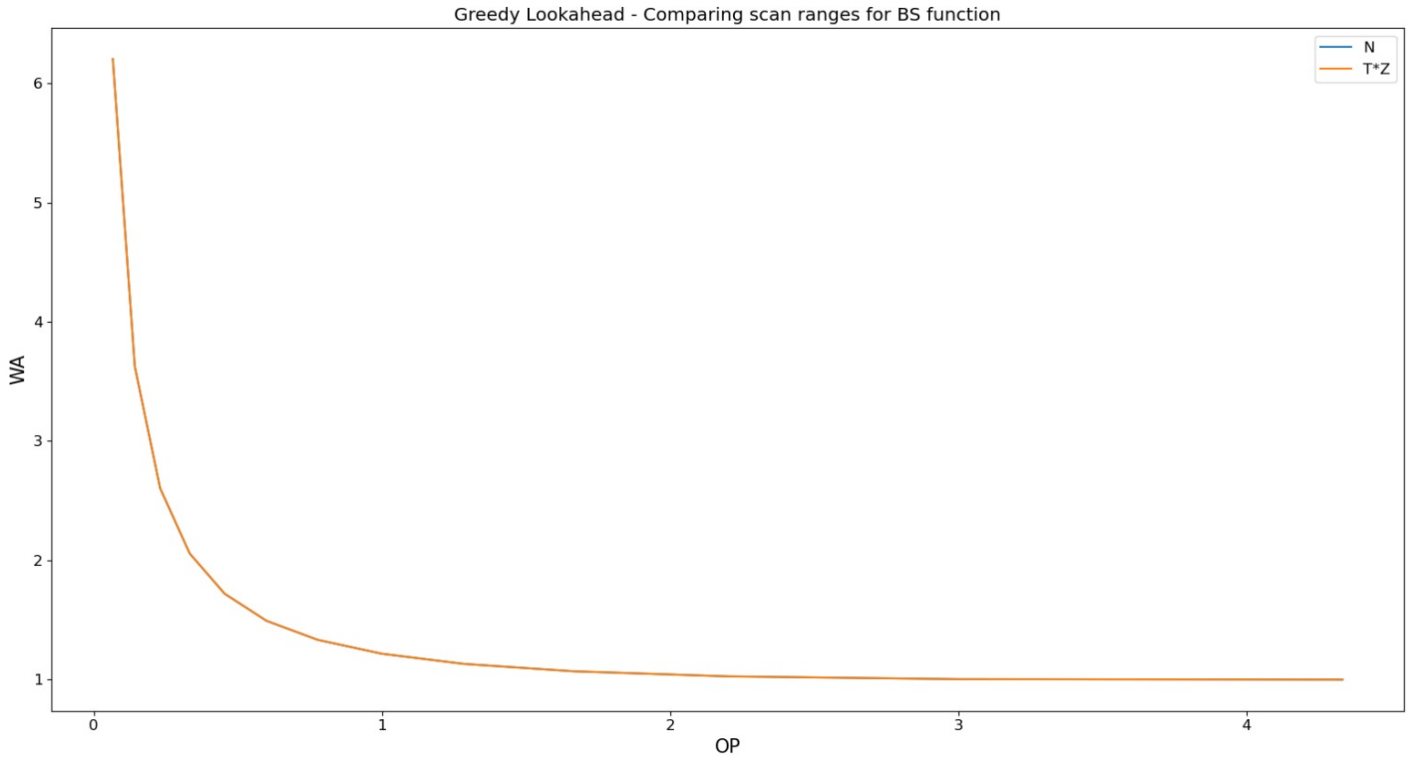
ניסוי מס' 5 – השוואת טווח הסריקה עבור הפונקציה BS

- תיאור הניסוי – נבצע השוואה של תוצאות הרצת אלגוריתם Greedy Lookahead כאשר בפונקציית ה-BS אנחנו מבצעים סריקה עתידית של $T \cdot Z$ דפים.

- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. כל ערך של WA נלקח כממוצע של 10 הרצות. בנוסף, בניסוי זה השתמשנו בפונקציית BS המעודכנת לפי תוצאות ניסוי מס׳ 4.
- תוצאות הניסוי – נציג את תוצאות הניסוי בטבלה ובגרף:

SCAN RANGE / OP	N	$T \cdot Z$
0.0666	6.20362	6.2022
0.1428	3.62675	3.62689
0.2307	2.60431	2.60573
0.3333	2.05763	2.05797
0.4545	1.71999	1.72097
0.6	1.49258	1.49311
0.777	1.33301	1.33355
1	1.2169	1.21678
1.285	1.13152	1.13118
1.666	1.06942	1.06934
2.2	1.02719	1.02677
3	1.00342	1.00399
4.333	1	1

נציג גם את התוצאות באמצעות גרף, המציג את ביצועי האלגוריתם עבור כל אחד מטווחי החיפוש:



ניתן לראות כי צמצום טווח החיפוש אינו פוגע כמעט כלל בביצועי האלגוריתם (למעשה שני הגרפים ממש מתלכדים). תוצאה זו מאששת את ההשערה שלנו לפיה עלינו להתמקד רק בעתיד "הקרוב" בעת חישוב הניקוד עבור בלוק מסוים. ניתן לראות בגרף כי התוצאות למעשה נותרות זהות. נוכל אם כך לצמצם את טווח הסריקה בפונקציית ה-BS ובכך הצלחנו להשיג את שתי המטרות העיקריות שלנו בחלק זה – שיפור הסיבוכיות ושמירה על הביצועים.

פונקציית Block score בגרסתה הסופית:

Define: $blockScore(B, ws, i)$:

1. Define $blockScore = 0$.
2. Define $pagesInBlock = \{p \in [0, U \cdot Z - 1] \mid p \in B \text{ and } p \text{ is valid}\}$.
3. $for(long\ long\ pos = i; pos < i + T \cdot Z \text{ and } pos < N;\ pos++)$:
   3.1. if $ws[pos] \in pagesInBlock$:
      3.1.1. $pagesInBlock \leftarrow pagesInBlock \setminus \{ws[pos]\}$.
      3.1.2. if $|pagesInBlock| = 0$ – return $blockScore$.

3.2. $blockScore += \frac{|pagesInBlock|}{(pos-i+1)^{\alpha}}$.

4. Return $blockScore$.

הגדלת מרחב החיפוש

בחלק הקודם עסקנו בפונקציית ה-Block score. פונקציה זו נותנת "דירוג" לכל בלוק אשר עוזר לנו לבחור את הבלוק האידיאלי למחיקה. כעת נרצה להסתכל על הבעיה מזווית שונה.

תזכורת: אנחנו מסמנים באות Y את ערך ה-minValid. בהינתן כל הבלוקים הפיזיים, ערך ה-minValid מתאר את המס' המינימלי של דפים ולידיים בבלוק פיזי כלשהו בזיכרון.

בנוסף הגדרנו את V להיות מערך של מצביעים בגודל $Z+1$. לכל $0 \leq i \leq Z$ הגדרנו את $V[i]$ להיות הקבוצה הבאה:

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

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

בעת הרצת האלגוריתמים Greedy GC ו-Greedy Lookahead, בכל פעם שאנחנו נדרשים לבצע מחיקה של בלוק אנחנו מסתכלים על הקבוצה $V[Y]$. קבוצה זו מכילה את הבלוקים "המועמדים" למחיקה. את בחירת הבלוק הספציפי למחיקה ביצענו בדרך שונה כתלות באלגוריתם:

- Greedy GC בוחר אקראית את אחד הבלוקים מתוך הקבוצה $V[Y]$.
- Greedy Lookahead בוחר בלוק למחיקה לפי הדירוג המתקבל מפונקציית ה-Block score.

באלגוריתם Greedy GC אין לנו מידע על עתיד הכתיבות, ולכן הבחירה מתוך הקבוצה $V[Y]$ היא אכן אופטימלית. עם זאת, בהינתן שידוע לנו עתיד הכתיבות, אין הכרח שהבחירה מתוך $V[Y]$ בהכרח תהיה אופטימלית.

דוגמה מס' 4: נסתכל על מקרה פשוט אשר ידגים את הסוגייה עמה אנחנו מתמודדים. נניח כי הקבוצות $V[Y]$ ו-$V[Y+1]$ מכילות כל אחד בלוק בודד. בה"כ הבלוקים נראים כך:

Block no. 1	Block no. 2
1	4
2	5
3	X
X	X
X	X
X	X

מתקיים: $V[Y] = \{2\}, V[Y+1] = \{1\}$. כעת אנחנו מגיעים למצב בו עלינו לבחור בלוק למחיקה, כאשר נתון לנו רצף הכתיבות העתידיות (החל מהכתיבה הנוכחית הממתינה ל-GC):

$$ws = [\ldots,4,5,6,7,8,\ldots]$$

במימוש הנוכחי של האלגוריתם Greedy Lookahead, הבלוק אשר ייבחר למחיקה הוא בלוק מס' 2 מכיוון שהוא הבלוק היחיד שמועמד למחיקה. אם נבצע את המחיקה ונמשיך לכתוב את הדפים הבאים לפי הסדר נגיע למצב הבא:

Block no. 1	Block no. 2
1	X
2	X
3	4
X	5
X	6
X	7

כעת, לפני שנוכל לכתוב את דף מס' 8 ניאלץ לבצע שוב GC, ונבחר למחיקה את בלוק מס' 1. מחיקה זו תעלה לנו בהעתקה של 3 דפים. מצב הזיכרון אחרי מחזור ה-GC השני והכתיבה של דף מס' 8:

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

מנגד, אם היינו בוחרים דווקא את בלוק מס׳ 1 עבור מחזור ה-GC הראשון וממשיכים את הכתיבות לפי הסדר, היינו מגיעים למצב הבא לפני הכתיבה של דף מס׳ 7:

Block no. 1	Block no. 2
1	X
2	X
3	X
4	X
5	X
6	X

אמנם אנחנו נאלצים לבצע GC איטרציה אחת **מוקדם** יותר, אך נשים לב שכעת אנחנו מוחקים בלוק בו כל הדפים מתים, כלומר אנחנו לא משלמים על העתקת דפים כלל ולא פוגעים ב-WA. מצב הזיכרון אחרי מחזור ה-GC השני והכתיבה של דף מס׳ 8:

Block no. 1	Block no. 2
1	7
2	8
3	
4	
5	
6	

קיבלנו שיפור ב-WA ובנוסף לוקליות טובה יותר בזיכרון (דפים חיים שמקובצים יחד – נדון על כך בהרחבה בפרק הבא).

ניתן לראות אם כך כי הרחבת מרחב החיפוש מאפשרת לנו לפעמים לבחור בלוקים משתלמים יותר למחיקה. נוכל להציע אם כך הרחבה לאלגוריתם Greedy Lookahead, אשר ירשה לעצמו להסתכל על בלוקים הנמצאים גם בקבוצות $V[X]$ עבור $Y < X$. כל בלוק כזה ידורג לפי פונקציית ה-BS והבלוק בעל ציון ה-BS הגבוה ביותר **מבין כל הבלוקים המועמדים** ייבחר למחיקה.

הרחבה זו של האלגוריתם מעלה מס' בעיות ו-Tradeoffs שיש לדון בהם:

1. הרחבת מרחב החיפוש מעלה את סיבוכיות הזמן של האלגוריתם. בממוצע, תחת הנחת פיזור אחיד מתקיים כי: $Y = \alpha' Z$, כאשר $\alpha' = \frac{EF-1}{EF} = \frac{EZ-L}{EZ}$. בנוסף, דנו בחלק הקודם בפרק זה בסיבוכיות הריצה של הפונקציה Block score. גם לאחר ביצוע אופטימיזציות לזמן הריצה של הפונקציה, היא עדיין עומדת על $O(T \cdot Z)$ במקרה הגרוע. ככל שנסרוק יותר איברים במערך V, סיבוכיות הזמן הכוללת תגדל בפקטורים פולינומיאליים נוספים. ברמה התיאורטית ניתן להתייחס לגדילה זו בצורה זניחה, אך ברמה הפרקטית יש לכך השלכות משמעותיות על ביצועים של מערכת שתרצה לנסות ולממש אלגוריתם שכזה כחלק מה-I/O Flow שלה. בנוסף, יש לבחון עד כמה פתרון זה אכן משפר את מדד ה-WA, ולשקול האם שיפור זה מצדיק את התשלום הנוסף בזמן.
2. חשוב לזכור כי בחירה של בלוק למחיקה שאינו מהקבוצה $V[Y]$ תוביל לכך שבמחזור ה-GC ה-i בהכרח נבצע איסוף שיפגע ב-WA בהשוואה לבחירה של בלוק מהקבוצה $V[Y]$, וזאת משום שבהכרח נידרש לבצע העתקה של יותר דפים. נרצה לעשות זאת רק אם יובטח לנו כי במחזור ה-GC ה-$i+1$ (ואולי גם במחזורים הבאים אחריו) נקבל תועלת שתפצה על העתקת הדפים המיותרת, לכאורה. כלומר, **התועלת העתידית צריכה להיות גבוהה מהעלות המיידית**. למרות שראינו בדוגמה מס' 5 מקרה בו תועלת זו אכן מושגת, פתרון הבעיה במקרה הכללי אינו פשוט כל כך, וייתכן מאד כי פונקציית ה-Block score אותה הגדרנו בחלק הקודם לא תתאים כעת באותו האופן.

ניסוי מס' 6 – הרחבת מרחב הבלוקים לבחירה עבור Greedy Lookahead

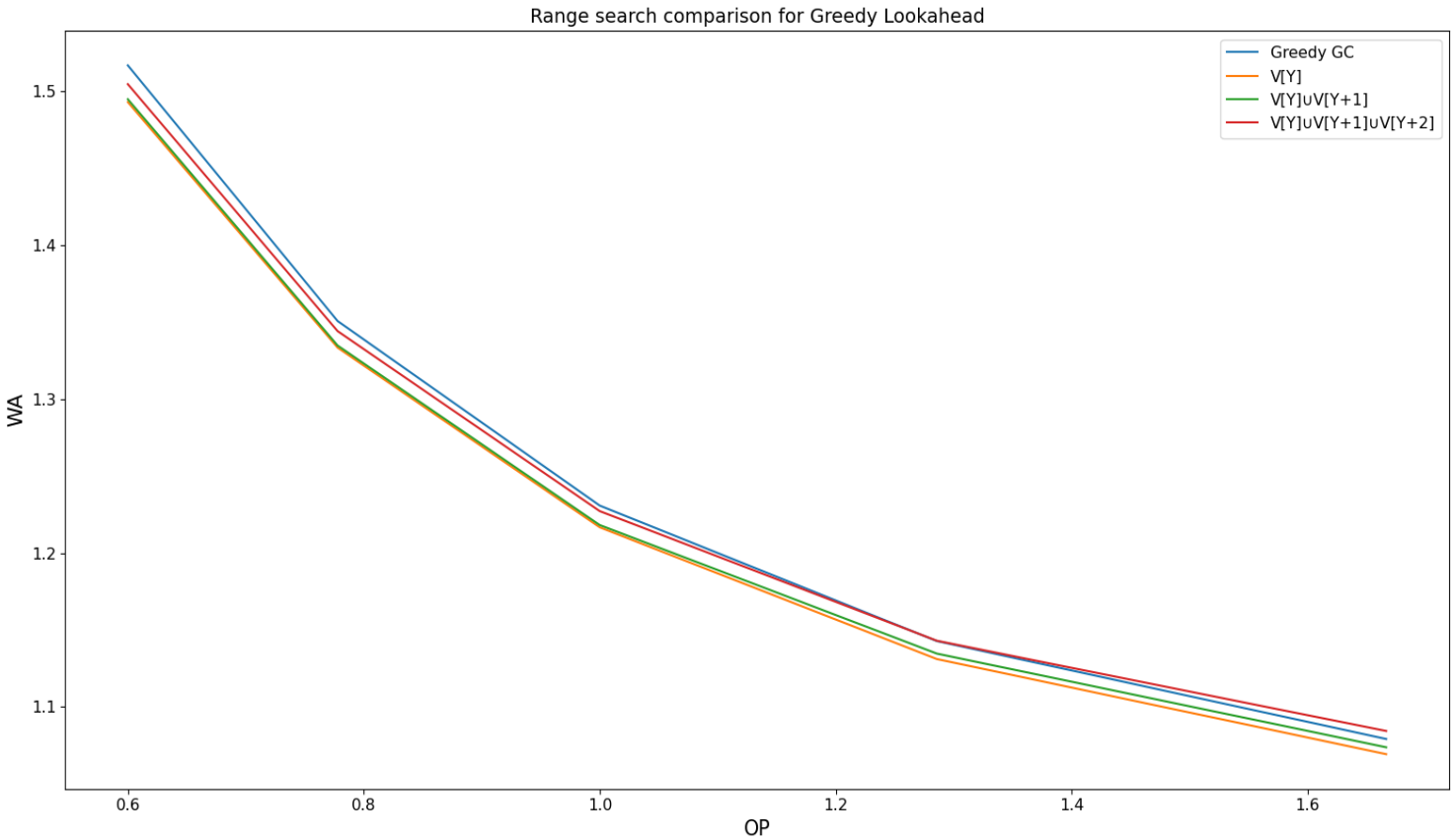
- תיאור הניסוי – נבצע השוואה של תוצאות הרצת אלגוריתם Greedy Lookahead כאשר אנחנו מאפשרים לבחור בלוק למחיקה מתוך טווח רחב יותר של בלוקים בהשוואה לאלגוריתם המקורי.
- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. כל ערך של WA נלקח כממוצע של 10 הרצות. נבצע השוואה בין 3 מרחבי חיפוש:
  1. $V[Y]$ (מרחב החיפוש המקורי)
  2. $V[Y] \cup V[Y+1]$

3. $V[Y] \cup V[Y+1] \cup V[Y+2]$

בנוסף, בניסוי זה השתמשנו בפונקציית BS המעודכנת לפי תוצאות ניסוי מס׳ 4 ובטווח החיפוש המעודכן לפי ניסוי מס׳ 5.

- תוצאות הניסוי – נציג את תוצאות הניסוי בטבלה ובגרף :

Range \ OP	0.066	0.142	0.230	0.333	0.454	0.6	0.777	1	1.285	1.666	2.2	3	4.333
$V[Y]$	6.202	3.626	2.605	2.057	1.720	1.493	1.333	1.216	1.131	1.069	1.026	1.003	1
$V[Y] \cup V[Y+1]$	6.264	3.636	2.611	2.059	1.723	1.494	1.334	1.218	1.134	1.073	1.037	1.007	1
$V[Y] \cup V[Y+1] \cup V[Y+2]$	6.617	3.716	2.644	2.080	1.736	1.504	1.344	1.227	1.143	1.084	1.051	1.012	1



מסקנות: ניתן לראות מתוצאות הניסוי כי התוצאות הטובות ביותר הושגו עבור מרחב החיפוש המקורי הכולל את הבלוקים בקבוצה $V[Y]$ בלבד. לכאורה ניתן לראות בכך סתירה לטענה כי הגדלת טווח החיפוש עשויה לשפר את ה-WA, אך ישנו הסבר נוסף אשר יכול ליישב את הדעת בנוגע לתוצאות הניסוי. יש לזכור שפונקציית ה-Block score בה אנחנו משתמשים מהווה **יוריסטיקה** אשר נועדה לנבא את התנהגות הזיכרון לאורך הזמן. בנוסף, את כל האופטימיזציות וכיוונון הפרמטרים שביצענו על הפונקציה הזו ביצענו עבור מרחב החיפוש המקורי אשר כלל את הקבוצה $V[Y]$ בלבד. בניסוי הזה השתמשנו באותה הפונקציה עבור הבלוקים בקבוצה $V[Y+1] \cup V[Y+2]$, ומהתוצאות אנחנו לומדים כי היוריסטיקה שלנו מתקשה לבצע הערכה מספיק טובה עבור הבלוקים אשר בהם יש יותר דפים חיים. בנוסף, אנחנו רואים כי גם השיפור התאורטי אשר ניתן להשיג כאן יחסית זניח, ולא שווה את המחיר הוודאי אשר ניאלץ לשלם בסיבוכיות הזמן של האלגוריתם.

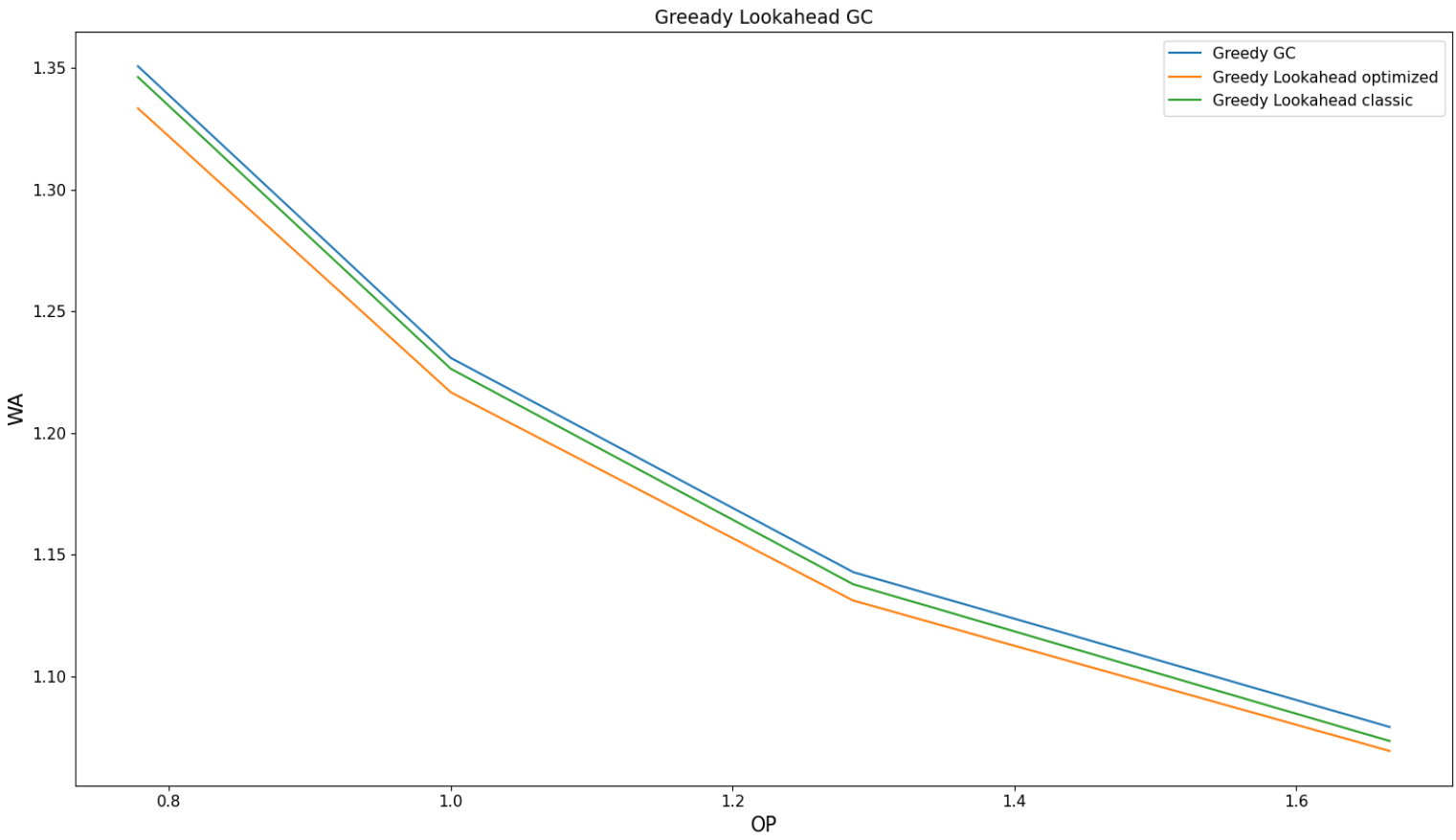
<u>כיוון להרחבה והמשך מחקר:</u>

פתרון אפשרי לבעיה עם היוריסטיקה הנוכחית היא לבצע התאמה נוספת לפונקציית ה-Block score, ולהוסיף לציון הבלוק משקלים אשר מייצגים את הקבוצה אליה הבלוק שייך ($V[Y]$, $V[Y+1]$ וכו'..). כלומר, ככל ש-X גדל, כך הבלוק השייך לקבוצה $V[X]$ יקבל משקל נמוך יותר (המשקל יוכפל בציון אשר קיבל הבלוק מפונקציית ה-Block score המקורית). כך נצליח (בתקווה) לפצות ולהבטיח שרק בלוקים עבורם התועלת מספיק גדולה ייבחרו מהקבוצות $V[X]$ עבור $Y < X$.

בנוסף, ניתן כמובן לנסות פונקציות Block score שונות אשר עשויות להתאים יותר לבעיה זו (לדוגמה פונקציות בעלות דעיכה אקספוננציאלית במקום דעיכה פולינומיאלית כפי שהשתמשנו עד כה).

ניסוי מס' 7 – השוואת ביצועי אלגוריתם Greedy Lookahead

- תיאור הניסוי – נבצע השוואה של תוצאות הרצת אלגוריתם Greedy Lookahead הקלאסי (ללא אופטימיזציות כלל) אל מול הרצת האלגוריתם המשלב את כל האופטימיזציות שהוצגו בחלקים הקודמים.
- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. כל ערך של WA נלקח כממוצע של 10 הרצות. בנוסף, בניסוי זה השתמשנו בפונקציית BS המעודכנת לפי תוצאות ניסוי מס' 4 ובטווח החיפוש המעודכן לפי ניסוי מס' 5. בנוסף ממסקנות ניסוי מס' 6 נמשיך להשתמש בטווח החיפוש המקורי (הכולל את הקבוצה $V[Y]$ בלבד).
- תוצאות הניסוי – נציג את תוצאות הניסוי בגרף:



ניתן לראות בצורה גרפית כיצד השיפורים לאלגוריתם משפרים את ביצועי הכתיבה עבור כל ערכי ה-OP.

סיכום

בפרק זה הצגנו את אלגוריתם Greedy Lookahead. הגרסה הראשונית של האלגוריתם מהווה אופטימיזציה טבעית לאלגוריתם Greedy GC הקלאסי תחת הנחות המודל שלנו. עם זאת, הראינו כי שניתן לבצע אופטימיזציות נוספות אשר מביאות גם לשיפור בביצועי האלגוריתם ע״י הורדת ה-WA עבור כל ערכי ה-OP האפשריים, ובנוסף משפרות את סיבוכיות זמן הריצה של האלגוריתם בצורה משמעותית.

Generational GC

רעיון כללי ומוטיבציה

עד כה עסקנו באלגוריתמים ושיפורים אשר נוגעים לשלב ה-GC עצמו. כעת נרצה לבחון אפשרות לנצל את המודל וההנחות שלנו על מנת לשנות את אלגוריתם כתיבת הדפים לזיכרון. בהינתן שנתון לנו עתיד הכתיבות, נרצה לתכנן אלגוריתם אשר ימצא את הבלוק המתאים ביותר עבור הדף הנתון, בצורה כזו שתמזער את מספר מחיקות הבלוקים הכולל.

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:

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).

<u>טרמינולוגיה:</u>

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

<u>דוגמה מס' 5 – תוחלת חיים של דף:</u> נסתכל על רצף הכתיבות הבא 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

כאשר lpn (logical page number) מייצג את מס' הדף הלוגי.

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.

<u>דוגמה מס' 6:</u> 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:

$Block\ 0$	$Block\ 1$	$Block\ 2$	$Block\ 3$

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:

$Block\ 0$	$Block\ 1$	$Block\ 2$	$Block\ 3$

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

תיאור האלגוריתם

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). את

הבלוקים הייעודיים נכנה בשם generational blocks. בלוקים אלו יתמלאו בהדרגה, ובכל פעם שאחד מהם יתמלא ייבחר בלוק חדש להחליף את מקומו.

Suppose: ws of size N, where $ws[i]$ is the ith 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.

מימוש האלגוריתם

אלגוריתם Greedy Lookahead ממומש בסימולטור. הוראות ההפעלה נמצאות בקובץ README.md בדף ה-Github של הפרויקט.

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.

בהמשך הפרק נעסוק באופטימיזציות והרחבות לאלגוריתם ה-Generational GC, 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 את יעילותו כבר הוכחנו בפרק הקודם.

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 ***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.

ניסוי מס' 8 – אלגוריתם הכתיבות Generational

- תיאור הניסוי – הניסוי הראשון יהיה דומה לניסוי מס' 2 ויבחן את השינוי ב- write amplification כתלות ב-over-provisioning.
- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. כל ערך של WA נלקח כממוצע של 20 הרצות. בנוסף, בניסוי זה השתמשנו בפונקציית BS המעודכנת לפי תוצאות ניסוי מס' 4 ובטווח החיפוש

המעודכן לפי ניסוי מס' 5. בנוסף ממסקנות ניסוי מס' 6 נמשיך להשתמש בטווח החיפוש המקורי (הכולל את הקבוצה $V[Y]$ בלבד).

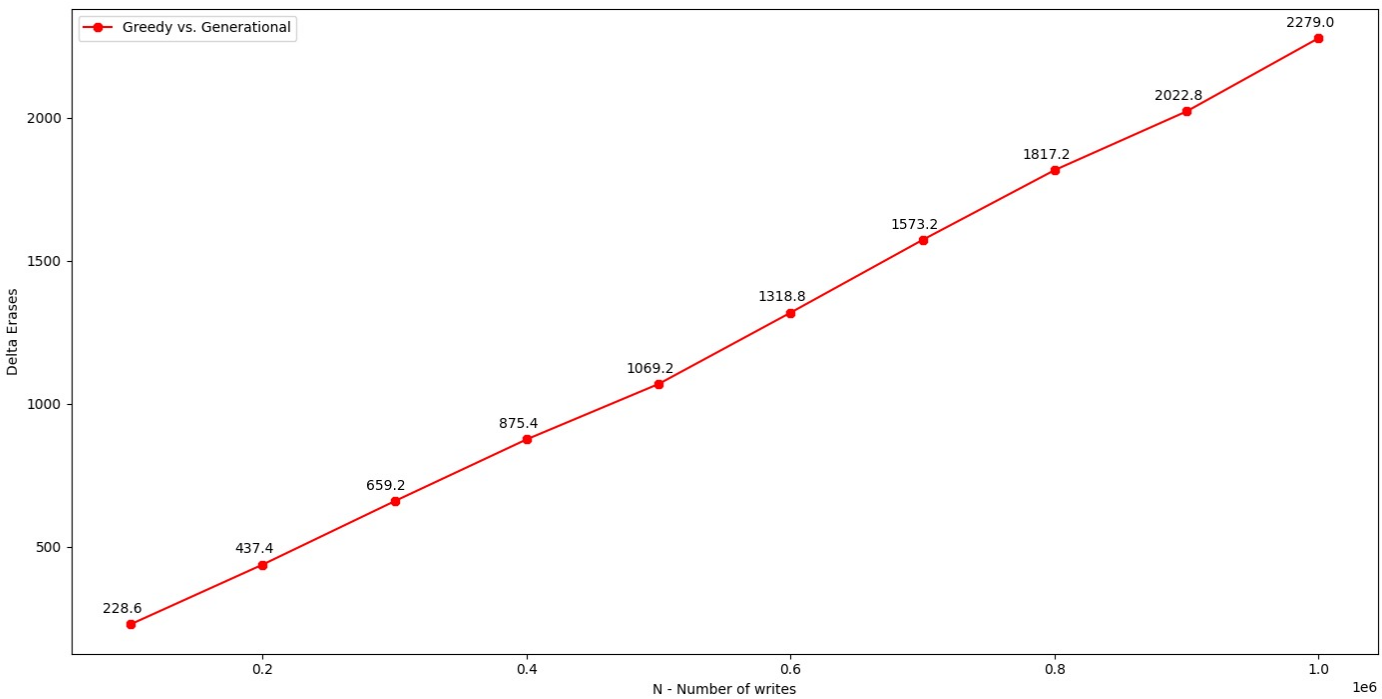
- תוצאות הניסוי – נציג את התוצאות בטבלה. שתי העמודות הראשונות מכילות את ערכי ה-WA עבור האלגוריתמים Greedy GC ו-Greedy Lookahead. התוצאות עבור Greedy Lookahead הן עבור הגרסה **הראשונית** של האלגוריתם. נוסיף כעת עמודה נוספת עם תוצאות הניסוי עבור אלגוריתם Generational GC:

ALGORITHM / OP	GREEDY GC	GREEDY LOOKAHEAD	GENERATIONAL
0.0666	6.78079	6.70131	6.7064
0.1428	3.81117	3.78392	3.7505
0.2307	2.69403	2.68005	2.63024
0.3333	2.1092	2.10211	1.95548
0.4545	1.75361	1.7487	1.54587
0.6	1.51698	1.51163	1.31394
0.777	1.35085	1.34639	1.17735
1	1.2309	1.22644	1.09381
1.285	1.14286	1.13786	1.04253
1.666	1.07919	1.07342	1.012
2.2	1.03464	1.02795	1.00076
3	1.00655	1.00345	1.00002
4.333	1	1	1

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.

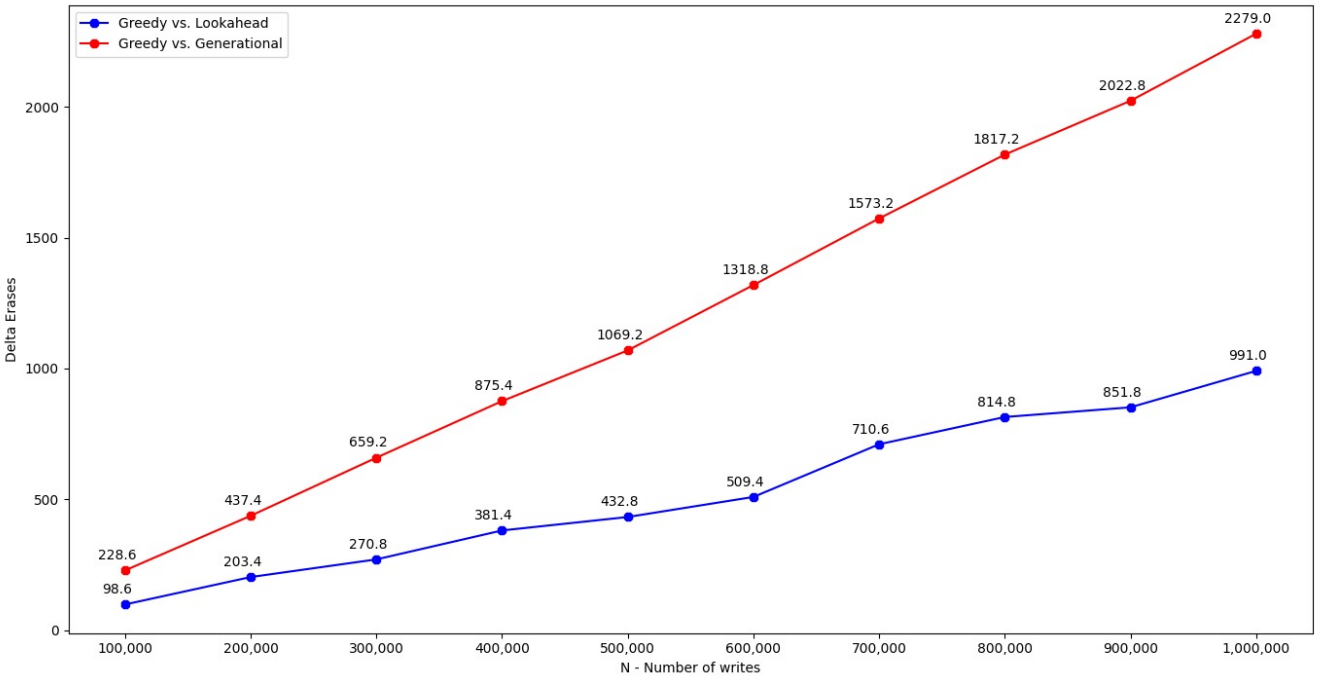
ניסוי מס' 9 – מדידת ΔE

- תיאור הניסוי – מדידת השינוי ב-$\Delta E = E_{Greedy} - E_{Generational}$ כתלות במספר הכתיבות N.
- תנאי הניסוי – $T = 64, U = 55, Z = 32$. הסימולציות הורצו עבור האלגוריתמים Greedy ו-Generational GC. 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 – נציג את תוצאות הניסוי 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.

אופטימיזציות והרחבות לאלגוריתם Generational

בחלק זה נרצה להרחיב את הדיון באלגוריתם ה-Generational ולנסות ולשפר את ביצועיו. על מנת להצליח במשימה זו נרצה להתמקד בסוגיה המרכזית והיא בחירת מס' הדורות עבור האלגוריתם. בתיאור האלגוריתם המקורי המוצג לעיל בחרנו להשתמש בשני דורות. הבחירה בשני דורות היא הבחירה הנאיבית והבסיסית ביותר. כפי שראינו בניסוי מס' 8 גם השימוש בשני דורות כבר מביא לשיפור בביצועים בהשוואה לשאר אלגוריתמי הכתיבה. אולם, עולה השאלה האם בחירה של מס' דורות גדול יותר יכול להביא לשיפור נוסף בביצועים?

באופן פורמלי נרצה לחקור את השאלה הבאה: **בהינתן הפרמטרים T, Z, U – מהו מס' הדורות אשר יביא לביצועים האופטימליים?**

התשובה לשאלה זו, כפי שיתברר בהמשך, די מורכבת ותלויה בלא מעט גורמים שנצטרך לקחת בחשבון. כדי לענות על השאלה נתכנן את האסטרטגיה שלנו להתמודדות עם הנושא:

1. תחילה, נרצה להבין את המשמעות של הגדלת מספר הדורות. נרצה להבין מהם היתרונות ומהם החסרונות בבחירת מס' דורות גדול אל מול מס' קטן יותר, ומהן המגבלות (ערכי מינימום ומקסימום) על מס' הדורות שניתן לבחור.
2. נערוך ניסויים בהם נרצה להבין מהי ההשפעה של כל אחד מהפרמטרים השונים T, Z, U על מס' הדורות האופטימלי, ומהי התלות בין הפרמטרים השונים
3. נתכנן יוריסטיקה אשר תנסה לגלם בתוכה את המסקנות התאורטיות והאמפיריות שקיבלנו, ונבחן את יעילותה

הרחבה אלגוריתם Generational לעבודה עם מס' דורות משתנה

כבסיס לעבודה בפרק זה עלינו לממש הרחבה של אלגוריתם Generational כך שיותאם לעבוד עם מס' דורות משתנה. נציג כעת את האלגוריתם המורחב $Generational(k)$, כאשר k הוא מס' הדורות:

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

Initial state:

1. Define: $Generations = \{0,1,\cdots,k-1\}$
2. $\forall i, 0 < i < k : \ Generations[i] = NIL$

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. $interval = \frac{Z \cdot U}{k}$
3. $bound \ = \ interval$
4. $for(int\ j = 0\ ;\ j \ < \ k-1;\ j++)$:
   4.1. if $pageScore < bound$:
      4.1.1. $return\ j$
   4.2. $bound \ += \ interval$
5. $return\ k-1$

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 $getGenBlock$ מחזירה את ה-generation block המתאים לפי פרמטר המסמל את מס' הדור. הפונקציה $updateGenBlock$ מקבלת פרמטר שאומר איזה דור צריך לעדכן וערך חדש לשים בתור ה-generation block הרלוונטי.

מימוש ההרחבה לאלגוריתם Generational

הרחבת אלגוריתם Generational לתמיכה במס' משתנה של generation blocks מומשה כחלק מהסימולטור. הוראות ההפעלה נמצאות בקובץ README.md בדף ה-Github של הפרויקט.

בחירת מספר הדורות

נרצה לדון כעת בשיקולים אשר צריכים להילקח בחשבון בעת בחירת מספר הדורות עבור אלגוריתם ה-Generational. כדי לעשות זאת נרצה להבין מהי המשמעות בהגדלה או הקטנת מספר ה- generation blocks.

כאשר אנחנו מגדירים k בלוקים בתור generation blocks, למעשה מעתה והלאה כל הכתיבות של דפים לוגים יופנו לאחד מבין k הבלוקים הללו. ברגע שאחד מהבלוקים מתמלא, הוא מוחלף בבלוק אחר מתוך ה-$freelist$, ובמידה וגם ה-$freelist$ מלא אנחנו מבצעים מחזור של GC על מנת לפנות מקום ואז בוחרים שוב בלוק מה-$freelist$ (שכעת מובטח לנו שיכיל בלוק פנוי). מכל האמור לעיל אנחנו מסיקים כמה מסקנות חשובות:

- כאשר אנחנו מבצעים GC, ה-generation blocks אינם נלקחים בחשבון. אם generation block מספר 1 מתמלא, ואין עוד בלוק ב-$freelist$, אנחנו נבצע מחזור של GC למרות שבכל שאר $k-1$ ה-generation blocks בהכרח **יש מקום פנוי לכתיבה של דפים**. כלומר באופן אפקטיבי אנחנו **מקטינים את מספר הבלוקים הפיזיים עבור GC מ-T ל-$T-k$** . ניתן

להסתכל על ה-generation blocks כעל בלוקים שהוצאו זמנית מהזיכרון, ומוכנסים אליו בחזרה לאחר שהם מתמלאים.

- בהמשך למסקנה הקודמת אנחנו יכולים כעת להסיק מהו הגבול העליון והתחתון עבור מס' הדורות k – מכיוון שכל generation block הוא למעשה בלוק אשר יוצא מהזיכרון עצמו ומשמש רק לכתיבות, אנחנו חייבים להבטיח כי בבלוקים הנותרים בזיכרון הפיזי יהיה לכל הפחות מקום לכתיבה של U דפים לוגיים. מכאן k יכול להיות לכל היותר כגודל היתירות של הזיכרון, כלומר $k \leq T - U$. בצד השני, עבור $k = 1$ נקבל למעשה את אלגוריתם Greedy GC המקורי (או Greedy Lookahead אם תהליך ה-GC משתמש באלגוריתם זה). מכאן שהגבולות עבור מס' הדורות הם $\boldsymbol{1 \leq k \leq T - U}$ (מגבלה זו תיאכף גם ע"י קוד הסימולטור אשר מריץ את האלגוריתם).

<u>תזכורת</u>: שתי האינווריאנטות אותן ננסה לשמר באמצעות אלגוריתם ה-Generational:

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.

כעת נוכל לנסח את המשמעות בהקטנה והגדלה של מס' הדורות:

- **ככל שמספר הדורות גדל**, אנחנו מרוויחים **לוקליות טובה יותר** של דפים בגילאים דומים אשר ייכתבו לאותם הבלוקים, ובכך משמרים את שתי האינווריאנטות. החיסרון הוא שהגדלה של מס' הדורות מביא **לירידה ביתירות האפקטיבית של הזיכרון**, ומורידה את הטריגר ל-GC. כלומר בפועל יתרחשו **יותר מחזורי GC** ובכך אנחנו מסתכנים בהעתקה של יותר דפים אשר עלול להביא לעלייה ב-WA.
- **ככל שמספר הדורות קטן**, אנחנו **מאבדים את הלוקליות** של דפים בגילאים דומים, מכיוון שטווח גדול יותר של גילאים ייכתב לכל generation block. בכך אנחנו פוגעים בשתי האינווריאנטות. היתרון הוא שהקטנה במס' הדורות **מעלה את היתירות של הזיכרון** ותביא **לפחות מחזורי GC** ובכך הסיכון לביצוע של העתקות דפים יורד.

ברור לנו כעת שקיימים tradeoffs ברורים בכל הקשור לבחירה של מס' הדורות. מס' הדורות האופטימלי הוא כזה אשר בו הלוקליות של הכתיבות מצליחה לפצות על הירידה ביתירות האפקטיבית של הזיכרון.

על מנת להצליח ולגבש אסטרטגיה לבחירה מושכלת של מס' הדורות עבור פרמטרים $\boldsymbol{T, Z, U}$ נתונים, נרצה להבין מהי ההשפעה (אם בכלל) של כל אחד מהפרמטרים על התוצאות המתקבלות. לשם כך נערוך מס' ניסויים אשר ייתנו לנו תמונה מקיפה על התלות וההשפעה של כל אחד מהפרמטרים הנתונים על בחירת מס' הדורות האופטימלי.

ניסוי מס׳ 10 – השפעת U על מס׳ הדורות האופטימלי

- תיאור הניסוי – מציאת מס׳ הדורות האופטימלי עבור אלגוריתם Generational, כאשר אנחנו מקבעים את הפרמטרים T, Z ומשנים רק את הפרמטר U.
- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. ערכי U נעים בטווח $12 \leq U \leq 60$ בקפיצות של 4. כל ערך של WA נלקח כממוצע של 10 הרצות. הסימולציות הורצו עבור האלגוריתם Generational, הכולל GC בעזרת אלגוריתם Greedy Lookahead עם השיפורים המתוארים בפרק הקודם. עבור כל ערך של U בוצעו ניסויים עבור מס׳ דורות בטווח $2 \leq k \leq 10$ (בכפוף למגבלה $k \leq T - U$).
- תוצאות הניסוי – תוצאות הניסוי מצורפות בנספח א׳.

כפי שניתן לראות מתוצאות הניסוי, מס׳ הדורות האופטימלי משתנה כתלות בפרמטר U. עובדה זו מתיישבת בצורה טובה עם התאוריה שלנו. נצפה שעבור ערכי OP שונים מס׳ הדורות האופטימלי ישתנה גם כן, וזאת מכיוון שאנחנו משנים את מס׳ הגילאים השונים עבור דפים בזיכרון, ולכן כמות הדורות האופטימלית תשתנה בהתאם.

ניסוי מס׳ 11 – השפעת T על מס׳ הדורות האופטימלי

- תיאור הניסוי – מציאת מס׳ הדורות האופטימלי עבור אלגוריתם Generational, כאשר אנחנו מקבעים את הפרמטר Z ואת ערכי ה-OP ומשנים רק את הפרמטר T.
- תנאי הניסוי – $Z = 32, N = 10^5$. בניסוי זה ננסה 4 ערכי T שונים: $T \in \{64,96,128,160\}$ ובנוסף ערכי U משתנים כך שיווצרו 13 ערכי OP **זהים** עבור כל ערך של T. כל ערך של WA נלקח כממוצע של 10 הרצות. הסימולציות הורצו עבור האלגוריתם Generational, הכולל GC בעזרת אלגוריתם Greedy Lookahead עם השיפורים המתוארים בפרק הקודם. עבור כל ערך של U (כלומר OP כלשהו) בוצעו ניסויים עבור מס׳ דורות בטווח $2 \leq k \leq 20$ (בכפוף למגבלה $k \leq T - U$).
- תוצאות הניסוי – תוצאות הניסוי מצורפות בנספח א׳. התוצאות מכילות 4 טבלאות (אחת עבור כל ערך של T).

כפי שניתן לראות מתוצאות הניסוי, גם לפרמטר T השפעה על מס׳ הדורות האופטימלי. כצפוי, ככל ש-T גדל כך מס׳ הדורות האופטימלי הממוצע גדל גם כן. תוצאה זו אכן מתיישבת בצורה טובה עם המודל התיאורטי שלנו, שכן ככל שישנם יותר דפים בזיכרון אנחנו ״מרוויחים״ יותר מהפרדה למס׳ גדול של דורות אשר מאפשר לנו לוקליות טובה, תוך שמירה על יתירות מקסימלית. ניתן לראות כי הגידול במספר הדורות האופטימלי אינו לינארי ביחס לגדילה בערכי T. לדוגמה – בהשוואה בין $T = 64$ ל-

$T = 128$, הערך האופטימלי הממוצע גדל פי 1.5 לערך, וזאת למרות שהערך של T גדל פי 2. מכאן אנחנו מסיקים שהזיכרון מאד רגיש לירידה ביתירות. בנוסף אנחנו רואים כי הגידול בערכי T משפיע על תוצאות האופטימום רק כאשר מגדילים אותו בצורה משמעותית.

ניסוי מס' 12 – השפעת Z על מס' הדורות האופטימלי

- תיאור הניסוי – מציאת מס' הדורות האופטימלי עבור אלגוריתם Generational, כאשר אנחנו מקבעים את הפרמטר T ואת ערכי ה-OP ומשנים רק את הפרמטר Z.
- תנאי הניסוי – $N = 10^5$, $T = 128$. בניסוי זה ננסה 3 ערכי Z שונים: $Z \in \{32,64,128\}$ ובנוסף ערכי U משתנים כך שיווצרו 13 ערכי OP **זהים** עבור כל ערך של Z. כל ערך של WA נלקח כממוצע של 10 הרצות. הסימולציות הורצו עבור האלגוריתם Generational, הכולל GC בעזרת אלגוריתם Greedy Lookahead עם השיפורים המתוארים בפרק הקודם. עבור כל ערך של U (כלומר OP כלשהו) בוצעו ניסויים עבור מס' דורות בטווח $2 \leq k \leq 20$ (בכפוף למגבלה $k \leq T - U$).
- תוצאות הניסוי – תוצאות הניסוי מצורפות בנספח א'. התוצאות מכילות 3 טבלאות (אחת עבור כל ערך של Z).

ניתן לראות כי ההשפעה של הפרמטר Z על ממוצע מס' הדורות האופטימלי היא יחסית זניחה ביחס לניסויים הקודמים.

Overloading Factor Heuristic

מניסויים 10-12 אנחנו רואים כי אכן ישנה השפעה של הפרמטרים השונים על מס' הדורות האופטימלי, אך גם אנחנו רואים כי עיקר ההבדלים נגרמים כתוצאה מהשינויים ב-OP (כלומר השינוי במס' הבלוקים הלוגיים U), ובנוסף אנחנו רואים כי גידול במס' הבלוקים הפיזי לא מביא למגמת עלייה ניכרת במס' הדורות האופטימלי. הסיבה לכך טמונה בחסרונות אשר תיארנו לעיל בהגדלת מס' הדורות – הקטנת היתירות של הזיכרון אשר גורמת לכך שאנחנו מבצעים יותר מחזורי GC ויותר העתקות של דפים, אשר בשלב מסוים מאפילים על היתרון היחסי אשר אנחנו משיגים בזכות החלוקה לדורות.

כפי שניתן לראות מתוצאות הניסויים והאנליזות שביצענו לאורך הפרק, הבעיה הכללית של מציאת מס' הדור האופטימלי לכל קומבינציה של הפרמטרים T, Z, U היא בעיה קשה מאד. עם זאת, לאחר למידת המסקנות מהניסויים שהרצנו, נרצה להציע יוריסטיקה אשר תאפשר לנו לבחור בצורה מושכלת את מס' הדורות האופטימלי עבור סימולציה נתונה.

דגשים עבור בחירת היוריסטיקה:

- היוריסטיקה צריכה להיות פשוטה ומהירה לחישוב ! לא נרצה לשלם בסיבוכיות זמן נוספת עבור חישובים מסובכים.
- נעדיף יוריסטיקה אשר תיתן ביצועים טובים עבור המקרה הממוצע ואולי תדייק פחות במקרי קיצון מאשר יוריסטיקה אשר תיתן ביצועים בינוניים עבור כל המקרים.

נתאר כעת את האינטואיציה מאחורי בחירת היוריסטיקה שלנו: נניח כי נתונים לנו הפרמטרים T, U, Z, ובנוסף נניח כי על ידי ביצוע ניסויים אמפיריים הגענו למסקנה כי מס' הדורות האופטימלי עבור שלושת הפרמטרים הנתונים הינו k, כך ש- $1 \leq k \leq T - U$.

נרצה לבחון את פיזור הדפים הלוגיים על פני ה-generation blocks השונים. יש לנו בסה"כ $U \cdot Z$ דפים לוגיים. האלגוריתם שלנו בוחר אינטרוולים שווים אשר מפרידים בין כל generational block. נניח לשם הפשטות כי ישנו פיזור שווה של הגילאים השונים על פני ה-generational blocks השונים. המשמעות היא שמס' הגילאים השונים אשר "משויכים" לכל generational block הוא $\frac{U \cdot Z}{k}$. המס' הזה מייצג את כמות הדפים הלוגיים אשר ייכתבו ל-generational block ספציפי. נעיר כי ההנחה שאנחנו מבצעים כאן לא מדויקת. למעשה התפלגות הכתיבות האחידה נותנת לנו הבטחה הסתברותית לגבי הגיל הממוצע של דף נתון. במסגרת פרויקט זה לא נרחיב את יותר בכל הקשור לכוונון האינטרוולים בהם אנחנו משתמשים עבור כל generation block, ולכן נשתמש בהנחה לפי העומס הנופל על כל generational block הוא אחיד.

<u>כיוון להרחבה והמשך מחקר:</u>

על מנת שנוכל להניח הנחת הפיזור האחיד על פני ה-generation blocks השונים, יש לבצע כיוונון של גדלי האינטרוולים עבור השיוך של דף מסוים ל-generational block. הכוונה היא לכוונון האינטרוולים בהם אנחנו משתמשים בפונקציה $getGen$, והתאמה של האינטרוולים להתפלגות הכתיבות הנתונה עבור הסימולציה. עבור התפלגות כתיבות יוניפורמית, היינו רוצים לבחור אינטרוולים אשר יביאו לאיזון טוב יותר של העומס על כל generational block. נוכל להשיג מטרה זו ע"י הגדלת כמות הגילאים "הצעירים" אשר יישלחו לבלוק generational הראשון, והקטנה של כמות הגילאים הנשלחים ל-generation blocks הבאים ככל שמתקרבים לגילאים השווים ל-$U \cdot Z$ (אלו האזורים הצפופים יותר מבחינת כתיבות ולכן נרצה לרווח אותם על פני יותר generation blocks כדי לשפר את הלוקליות של הכתיבות).

נחזור לתיאור היוריסטיקה – בהינתן שיש לנו $\frac{U \cdot Z}{k}$ דפים אשר נכתבים לבלוק כלשהו אשר מכיל Z דפים, נרצה לקבל מדד אשר מעריך את העומס אשר "ייפול" על אותו הבלוק. אם נחלק את מס' הדפים אשר נכתבים לבלוק במס' הדפים בבלוק, נוכל לקבל הערכה גסה לעומס אשר נופל על הבלוק. עבור המקרה שלנו נקבל כי העומס היחסי על כל generation block הינו $\frac{\frac{U \cdot Z}{k}}{Z} = \frac{U}{k}$.

<u>הגדרה</u>: בהינתן הפרמטרים T, U, Z, ה-Overloading factor (בקיצור – OF) עבור generation block יוגדר להיות $OF = \frac{U}{k}$, כאשר k הוא מס' הדורות האופטימלי עבור הפרמטרים הנתונים.

כאמור, המשמעות האינטואיטיבית של ה-OF הוא העומס הלוגי היחסי אשר נופל על generational block יחיד. ננסה להבין מה אנחנו מצפים מערך זה לקיים. אם $1 > OF$, המשמעות היא שהעומס הנופל על generational block קטן מגודל הבלוק. המשמעות היא שהיתירות שלקחנו קטנה מידי וגורמת לכך שאין ניצול טוב של ה-generational blocks. אם $OF = U$, המשמעות היא שכל העומס הלוגי נופל על ה-generational block. מצב זה יתאפשר רק במקרה בו $k = 1$, זהו למעשה מצב בו אלגוריתם Generational מתנוון לגמרי ולמעשה אנחנו חוזרים לעבוד לפי אלגוריתם Greedy GC הקלאסי.

נרצה להשתמש ב-OF על מנת להגדיר יוריסטיקה אשר תסייע לנו בבחירת מס' הדורות עבור הרצה נתונה.

<u>ניסוי מס' 13 – יוריסטיקת OF</u>

- תיאור הניסוי – מציאת ערך ה-OF הממוצע אשר ישמש אותנו להגדרת היוריסטיקה. בניסוי נדגום ערכי OF שונים המתקבלים מהרצת סימולציות, ולבסוף נחשב ערך OF ממוצע אשר ישמש להגדרת היוריסטיקה שלנו.
- תנאי הניסוי – $N = 10^5$, $Z = 32$. בניסוי זה נדגום 4 ערכי T שונים:

  $T \in \{64,96,128,160\}$. עבור כל צירוף של T, Z נבצע ניסוי תוך שימוש בערכי U משתנים. כל ערך של WA נלקח כממוצע של 10 הרצות. הסימולציות הורצו עבור האלגוריתם Generational, הכולל GC בעזרת אלגוריתם Greedy Lookahead עם השיפורים המתוארים בפרק הקודם. לכל צירוף של T, Z, U נחשב את k – מס' הדורות האופטימלי, וכן את הערך $OF = \frac{U}{k}$.
- תוצאות הניסוי – תוצאות הניסוי מצורפות בנספח א'. התוצאות מכילות 4 טבלאות עם ערכי k ו-OF עבור הניסוי הרלוונטי.

מתוצאות הניסוי ניתן לראות כי ערכי ה-OF השונים נשארים באופן יחסי דומים עבור הפרמטרים השונים. נשים לי כי עבור ערכי ה-OP מאד קטנים ערכי ה-OF גדלים משמעותית, זאת משום שערכי U גדלים וערכי k לרוב קטנים ומוגבלים על ידי החסם העליון $T - U$. בנוסף, נשים לב שעבור $OP > 1.5$ ערכי ה-WA שואפים ל-1, והתוצאות נשארות לרוב זהות בכל העמודות, עם הבדלים זניחים בין הפרמטרים השונים. כזכור, אחת המטרות שלנו בבחירת היוריסטיקה היא בחירה של יוריסטיקה שתהיה מוצלחת עבור המקרה "הממוצע", גם במחיר של אי דיוק עבור ערכי קצה. לכן, בחישוב

היוריסטיקה נזניח את ערכי הקצה (עבור ערכי OP גדולים מאד או קטנים מאד) ונתמקד בערכי OP בטווח $\frac{1}{7} \leq OP \leq 1$.

נרצה לחשב ערך OF ממוצע, אשר משקלל את כל ערכי ה-OF השונים אשר קיבלנו בניסוי. נעשה זאת ע״י חישוב ערך OF ממוצע לכל אחת מ-4 הטבלאות שקיבלנו בניסוי, ולאחר מכן חישוב ממוצע על ארבעת ערכי ה-OF המייצגים את כל אחת מהטבלאות. **התוצאה הסופית תהיה ערך OF אשר משקף את העומס הלוגי האופטימלי על generation block יחיד**.

עבור תוצאות ניסוי 13 נקבל שערך ה-OF המשוקלל המתקבל הוא: $\overline{\boldsymbol{OF}} = \mathbf{15.3792}$.

כעת נגדיר את יוריסטיקת Overloading factor:

נגדיר את $\tilde{k}$ להיות מס׳ הדורות האופטימלי של יוריסטיקת OF. בהינתן פרמטרים T, U, Z עבור סימולציה נתונה, מס׳ הדורות ייקבע להיות:

$$\tilde{k} = min\left\{T - U, \left\lfloor \frac{U}{\overline{OF}} \right\rfloor\right\}$$

הסבר – ראשית, הבחירה של $\tilde{k}$ חייבת לקיים את התנאים עבור בחירה של דור כלשהו. לכן תמיד נוודא כי הערך שנותנת היוריסטיקה אינו גבוה מ- $T - U$, ובמידה וכן נבחר את $\tilde{k}$ להיות $T - U$, משום שזהו הערך הקרוב ביותר לערך היוריסטי אשר מקיים את התנאים. במידה והערך היוריסטי אינו גבוה מ- $T - U$, נחזיר אותו וניקח ערך שלם תחתון כדי לקבל מס׳ דור שלם. הבחירה של ערך שלם תחתון נובעת מכך שתוצאות הניסויים שערכנו הראו לנו שנעדיף לרוב להיות שמרנים ולהגדיל את היתירות של הזיכרון.

מימוש יוריסטיקת Overloading factor

הרצת סימולציות של אלגוריתם Generational תוך שימוש ביוריסטיקת OF התווספה לסימולטור. על מנת להפעיל את היוריסטיקה יש להפעיל את אלגוריתם Generational ולבחור את מס׳ הדורות להיות 0. במקרה זה בחירת מס׳ הדורות עבור הסימולציה תיעשה בעזרת חישוב הפונקציה היוריסטית כפי שתוארה לעיל. הסימולטור יעדכן את המשתמש במספר הדורות שהיוריסטיקה בחרה, ולאחר מכן ימשיך בהרצת סימולציית הכתיבה כרגיל. הוראות נוספות ניתן למצוא בקובץ README.md בדף ה-Github של הפרויקט.

כעת, נרצה לבחון את ביצועי היוריסטיקה שלנו אל מול הגרסה הבסיסית של אלגוריתם Generational ובנוסף מול הגרסה האופטימלית (אשר בוחרת תמיד את מס׳ הדורות האופטימלי). נצפה כי השימוש ביוריסטיקה ישפר את הביצועים בהשוואה לגרסה הבסיסית, ונשאף שהביצועים של הפונקציה היוריסטית שלנו יתקרבו כמה שיותר אל הגרסה האופטימלית.

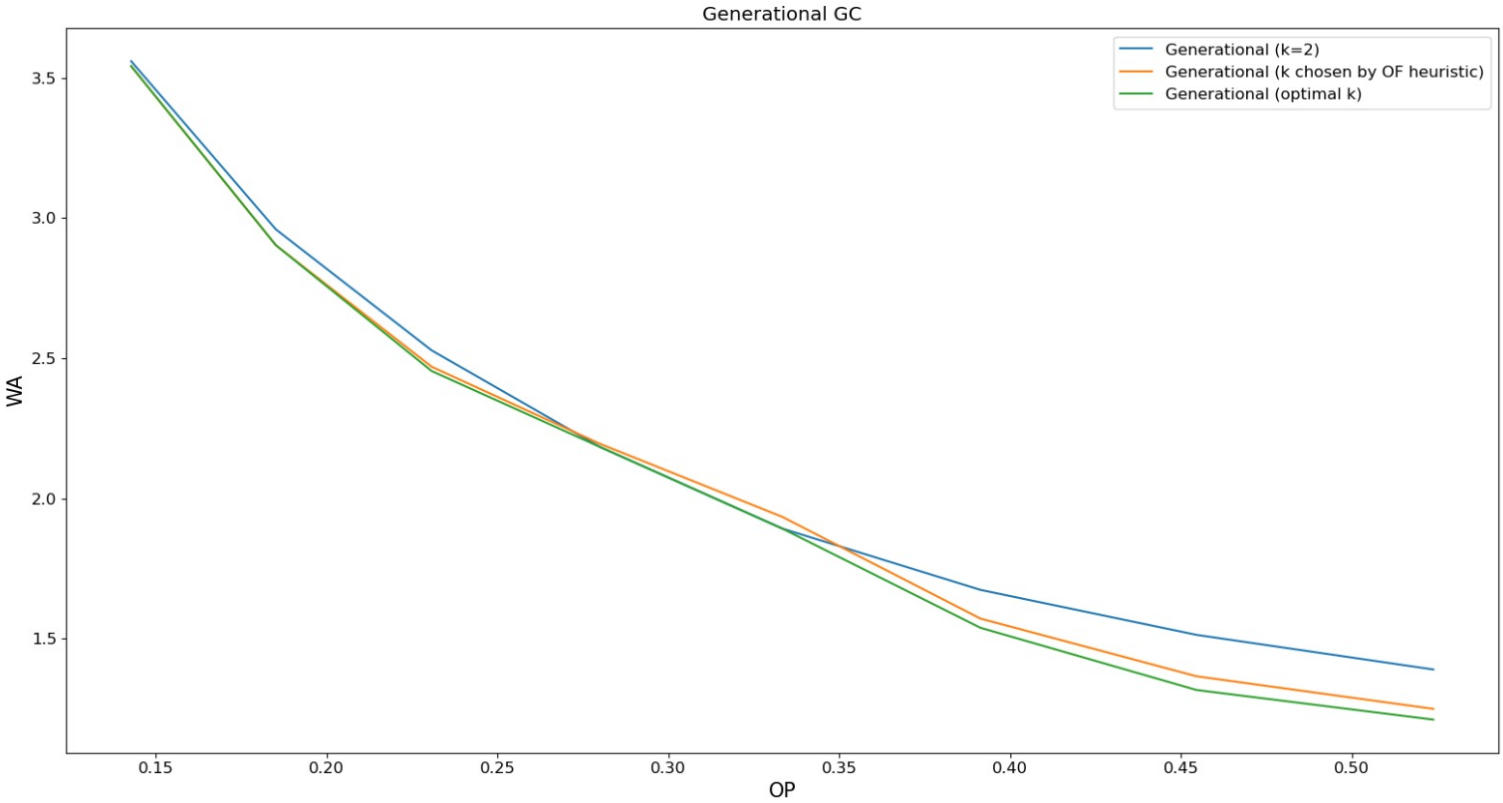
ניסוי מס' 14 – השוואת ביצועי אלגוריתם Generational

- תיאור הניסוי – השוואת ביצועי אלגוריתם Generational. נבצע שלוש הרצות של אלגוריתם Generational. ההרצה הראשונה תהיה עבור הגרסה הבסיסית של האלגוריתם המשתמשת תמיד בשני דורות. ההרצה השנייה תהיה בעזרת שימוש ביוריסטיקת OF. לכל ערך OP מספר הדורות ייבחר לפי הפונקציה היוריסטית. ההרצה השלישית תהיה תוך שימוש במס' הדורות האופטימלי לכל ערך OP.
- תנאי הניסוי – $T = 96, Z = 32, N = 10^5$. ערכי U נעים בטווח $42 \leq U \leq 90$ בקפיצות של 3. כל ערך של WA נלקח כממוצע של 10 הרצות. עבור GC נשתמש באלגוריתם Greedy Lookahead עם השיפורים המתוארים בפרק הקודם.
- תוצאות הניסוי – נציג את התוצאות בטבלה ובגרף. לכל ערך OP נציג את שלושת ערכי ה-WA עבור כל אחת מההרצות, ובנוסף נציג את מס' הדורות שבחרה היוריסטיקה אל מול מס' הדורות האופטימלי.

U	90	87	84	81	78	75	72	69	66	63	60	57	54	51	48	45	42
Generational WA (Basic)	6.154	4.491	3.559	2.959	2.528	2.184	1.893	1.674	1.513	1.39	1.295	1.222	1.165	1.121	1.087	1.059	1.038
Heuristic number of gens	5	5	5	5	5	4	4	4	4	4	3	3	3	3	3	2	2
Generational WA (OF)	6.171	4.488	3.542	2.903	2.469	2.201	1.934	1.572	1.366	1.25	1.214	1.155	1.111	1.077	1.053	1.059	1.038
OPT number of gens	2	3	5	5	7	2	2	6	7	7	7	8	8	9	8	8	8
Generational WA (OPT)	6.154	4.484	3.542	2.903	2.454	2.184	1.893	1.539	1.317	1.211	1.147	1.102	1.071	1.048	1.03	1.018	1.009

מקרא עבור הטבלה:

- השורה הראשונה מציגה את ערכי ה-WA עבור אלגוריתם Generational הבסיסי (המשתמש בשני דורות).
- השורה השנייה מציגה את מספר הדורות הנבחר בעזרת יוריסטיקת OF.
- השורה השלישית מציגה את ערכי ה-WA עבור אלגוריתם Generational אשר משתמש במספר הדורות שנבחר בעזרת יוריסטיקת OF.
- השורה הרביעית מציגה את מספר הדורות האופטימלי עבור אלגוריתם Generational. מספר זה חושב בעזרת ניסוי למציאת מס' הדורות האופטימלי (ערכים אלו נלקחו מתוצאות ניסוי מס' 11).
- השורה החמישית מציגה את ערכי ה-WA עבור אלגוריתם Generational אשר משתמש במספר הדורות האופטימלי.

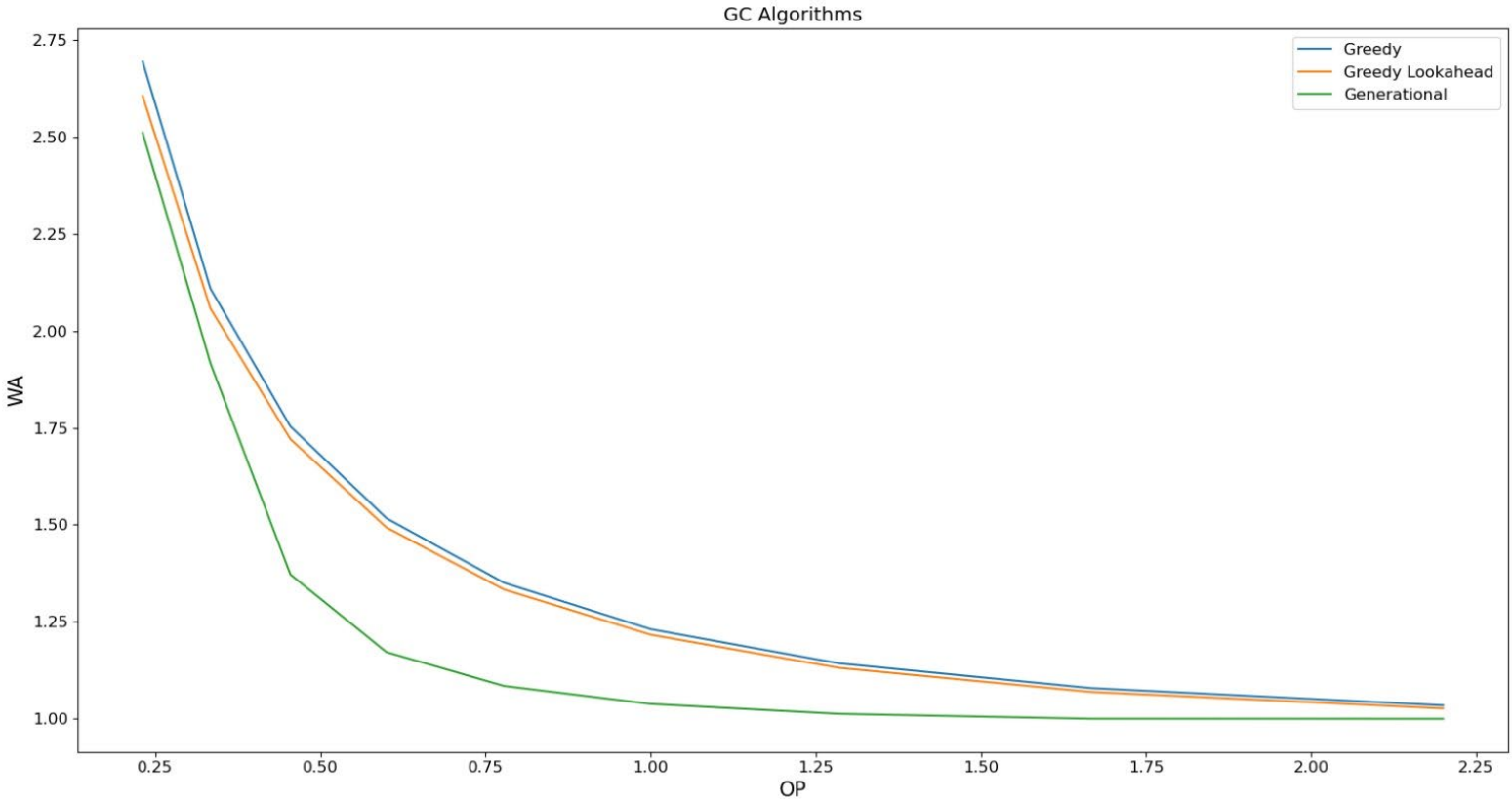


ניתן לראות כי היוריסטיקה שלנו אכן משיגה שיפור ביחס למקרה הבסיס עבור רוב ערכי ה-OP (הסימונים הצהובים בטבלה), ובנוסף עבור ערכי OP רבים אף מתקרבת לערך האופטימום. המקרים בהם היוריסטיקה "מפסידה" או משתווה למקרה הבסיס הם לרוב עבור ערכי הקצוות, שם צפינו כי היוריסטיקה שלנו תהיה פחות חזקה.

כעת, לאחר סיום העבודה על אלגוריתם Generational, נרצה לבצע השוואה סופית בין אלגוריתמי הכתיבה השונים בהם עסקנו בפרויקט זה.

ניסוי מס' 15 – השוואת ביצועי אלגוריתמי הכתיבה השונים

- תיאור הניסוי – השוואת ביצועי אלגוריתמי הכתיבה. ההשוואה תהיה בין אלגוריתם Greedy GC, Greedy Lookahead ו-Generational.
- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. ערכי U נעים בטווח $12 \leq U \leq 60$ בקפיצות של 4. כל ערך של WA נלקח כממוצע של 10 הרצות. אלגוריתם Greedy Lookahead הורץ עם כל השיפורים המתוארים בפרק הקודם. אלגוריתם Generational משתמש בערכי הדורות האופטימליים כפי שחושבו בניסויים 10-12, ובנוסף מבצע GC בעזרת Greedy Lookahead.
- תוצאות הניסוי – נציג את התוצאות בגרף (התוצאות המספריות של ניסוי זה הוצגו בנפרד בניסויים הקודמים עבור כל אחד מהאלגוריתמים):



ניתן לראות כי אלגוריתם Generational משיג את ביצועי הכתיבה הטובים ביותר עבור כל ערכי ה-OP.

סיכום

בפרק זה עסקנו באלגוריתם הכתיבה Generational. אלגוריתם זה ייחודי מכיוון שהוא מציג דרך חדשה לגמרי לביצוע הכתיבות לזיכרון. בנוסף, האלגוריתם משלב בתוכו את השיפורים שכבר השגנו בפרקים הקודמים, ובכך אנחנו מגיעים לביצועים הטובים ביותר עבור מודל ההנחות שלנו בפרויקט זה. ראינו כי הבחירה של מס׳ הדורות האופטימלי עבור פרמטרים שונים היא שאלה קשה, אך למרות זאת הצלחנו לבודד את ההשפעה של כל אחד מהפרמטרים ובכך לפתח יוריסטיקה אשר עזרה לנו להגיע לביצועים אשר מתקרבים לביצועים האופטימליים. במהלך העבודה גילינו כי יש עוד מקום רב להרחבות נוספות של אלגוריתם כתיבה זה, הן באופטימיזציות נוספות לאלגוריתם הכתיבה וחלוקת הכתיבות בין הדורות השונים, והן בבחירת יוריסטיקות נוספות אשר יכולות לשפר את הביצועים עבור סוגים שונים של מבני זיכרון.

התפלגות כתיבות Hot/Cold

בפרקים הקודמים עסקנו בפיתוח אלגוריתמים תחת הנחות של uniform writing, כלומר ההנחה היא שההסתברות לכתיבה של כל אחד מהדפים שווה. הנחה זו היא הנחה קריטית עבור המודלים התאורטיים עליהם מתבסס אלגוריתם Greedy GC. עם זאת, במערכות זיכרון אמיתיות הנחה זו כמובן לא מתקיימת. למעשה, עבור מערכות זיכרון אמיתיות לרוב משתמשים במודל כתיבות Hot/Cold.

תזכורת: Hot-Cold Writing – התפלגות כתיבה זו מדמה מצב בו יש לנו זיכרון המחולק לאזור של דפים חמים ואזור של דפים קרים. 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.

יש לציין כי תחת התפלגות הכתיבות Hot/Cold אלגוריתם Greedy GC אינו אופטימלי (בניגוד למודל הכתיבות היוניפורמי שם ניתן להוכיח את האופטימליות שלו).

בחלק זה נרצה להרחיב את יכולות הסימולטור שלנו לתמוך בהתפלגות כתיבות מסוג Hot/Cold, ולאחר מכן לערוך השוואה בין ביצועי האלגוריתמים השונים תחת התפלגות כתיבות זו.

מימוש מודל כתיבות Hot/Cold

הרצת סימולציות באמצעות מודל כתיבות Hot/Cold התווספה לסימולטור. ניתן להריץ את כל אחד מהאלגוריתמים השונים תוך שימוש בהתפלגות כתיבות זו. בעת בחירה בהתפלגות כתיבות Hot/Cold, יידרש המשתמש להזין את הפרמטרים r ו-p עבור הסימולציה. הוראות נוספות לעבודה עם התפלגות כתיבות זו ניתן למצוא בקובץ README.md בדף ה-Github של הפרויקט.

השוואה בין אלגוריתמים

נרצה כעת לערוך השוואה בין ביצועי האלגוריתמים השונים תחת התפלגות הכתיבות החדשה. שאר תנאי המודל זהים לאלו שהוצגו בפרק המבוא.

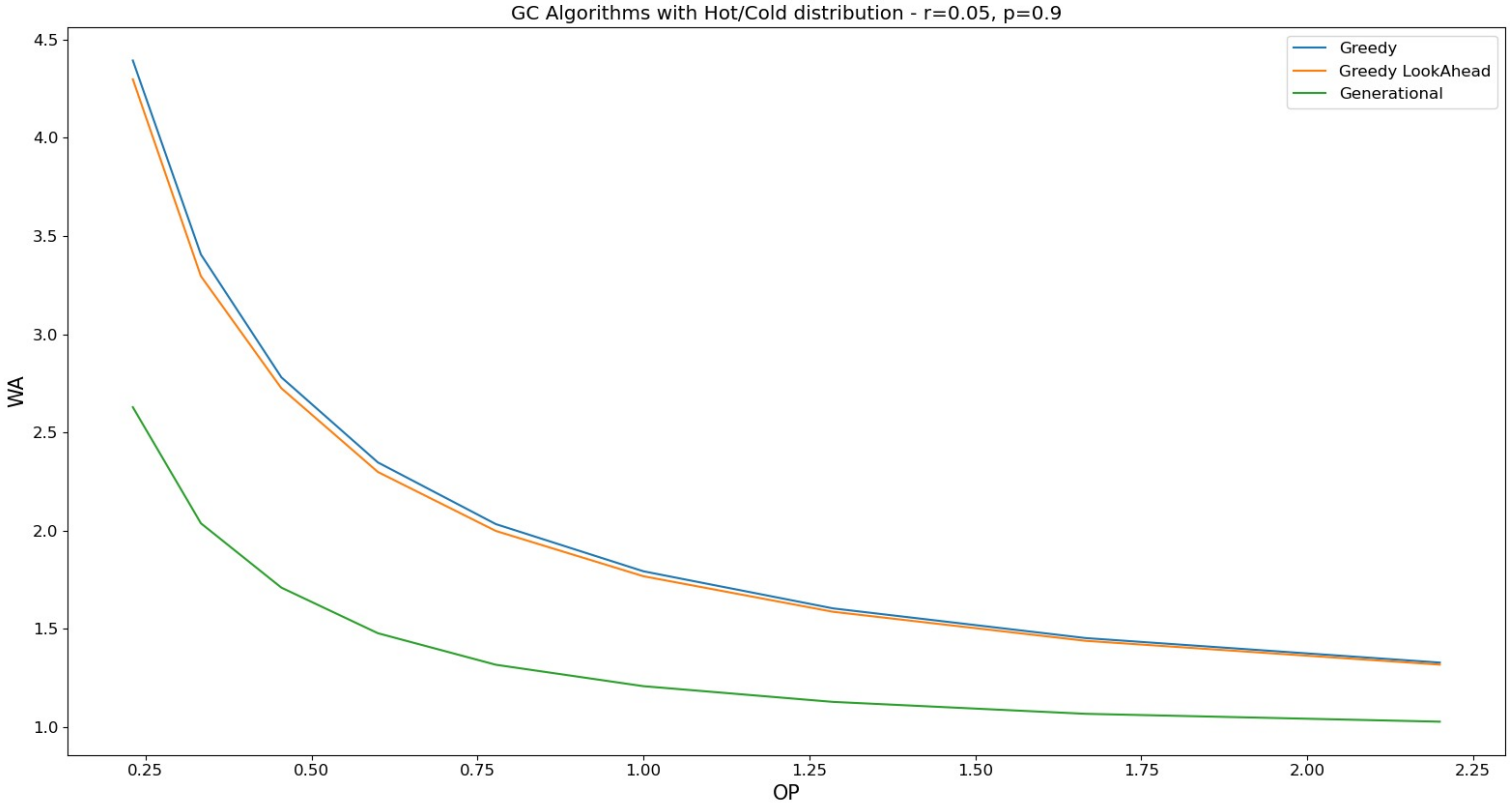
ניסוי מס' 16 – Hot/Cold writing distribution

- תיאור הניסוי – נערוך השוואה בין האלגוריתמים Greedy GC, Greedy Lookahead, Generational כאשר התפלגות הכתיבות היא Hot/Cold, ונבחן את השינוי ב-write amplification כתלות ב-over-provisioning.

- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. ערכי U נעים בטווח $12 \leq U \leq 60$ בקפיצות של 4. כל מדידה בניסוי היא של ערך write amplification. המדידה נקבעה על ידי לקיחת ממוצע של 10 הרצות עבור הפרמטרים הרלוונטיים. אלגוריתם Greedy Lookahead הורץ עם השיפורים אשר תוארו בפרק 2. עבור האלגוריתם Generational הניסוי הורץ עם מס' דורות קבוע אשר נקבע להיות $k = 2$, כאשר GC בוצע בעזרת אלגוריתם Greedy Lookahead עם השיפורים המתוארים בפרק הקודם. הרצנו ניסוי זה פעמיים עבור ערכים שונים של r ו-p.
- תוצאות הניסוי – נציג את תוצאות ה-WA בטבלה ובגרף:

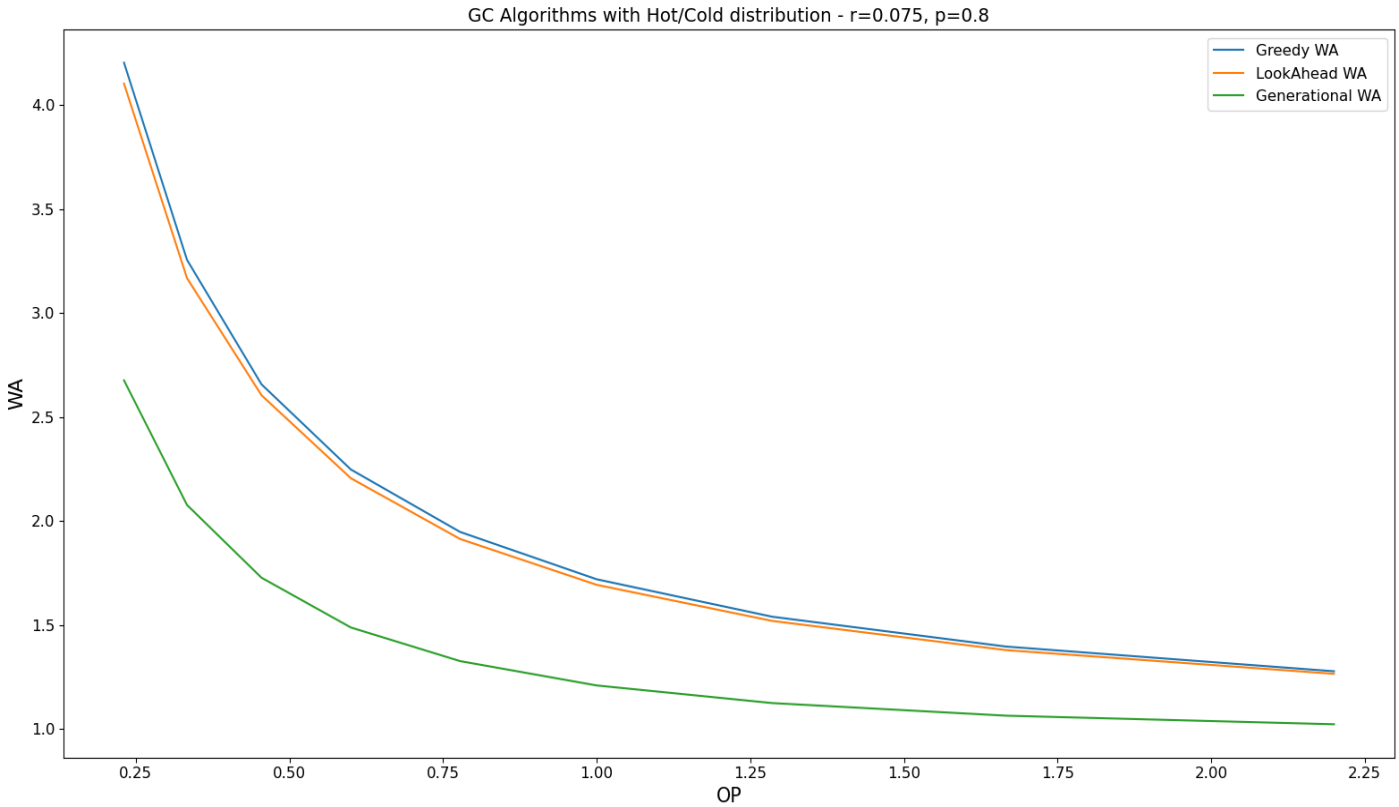
עבור $r = 0.05$ ו-$p = 0.9$ קיבלנו את התוצאות הבאות:

OP	0.066	0.142	0.230	0.333	0.454	0.6	0.777	1	1.285	1.666	2.2	3	4.33
Greedy	10.4184	6.18417	4.3936	3.4063	2.78102	2.34785	2.03381	1.79377	1.60495	1.45375	1.32898	1.22563	1.13838
Greedy Lookahead	9.93186	5.98079	4.29785	3.29575	2.72531	2.29921	1.99892	1.7686	1.58769	1.43969	1.31872	1.21722	1.13168
Generational	6.6379	3.73203	2.62901	2.03798	1.71056	1.47848	1.31794	1.20895	1.12897	1.06826	1.02824	1.00554	1



עבור $r = 0.075$ ו-$p = 0.8$ קיבלנו את התוצאות הבאות:

OP	0.066	0.142	0.230	0.333	0.454	0.6	0.777	1	1.285	1.666	2.2	3	4.33
Greedy	10.0915	5.94611	4.20256	3.25499	2.65679	2.24742	1.94731	1.71987	1.53978	1.3963	1.27766	1.18088	1.10029
Greedy Lookahead	9.33723	5.68783	4.10209	3.16697	2.60438	2.20537	1.91341	1.69321	1.51968	1.37927	1.26548	1.17071	1.09289
Generational	6.59106	3.78175	2.67588	2.07689	1.72731	1.48807	1.32646	1.20949	1.12478	1.06413	1.02291	1.00289	1



ניתן לראות כי עבור מודל כתיבות זה היתרון של אלגוריתם Generational בא לידי ביטוי בצורה משמעותית יותר בהשוואה להשוואה שביצענו תחת מודל הכתיבות הרגיל בניסוי מס' 15, וזאת למרות שהשתמשנו בגרסה הבסיסית של האלגוריתם שבוחרת תמיד בשני דורות בלבד!

עובדה זו לא מפתיעה, משום שתחת מודל הכתיבות Hot/Cold ישנה לוקליות הרבה יותר טובה של דפים – הגילאים של הדפים מתחלקים בצורה טובה מאד לשתי קבוצות גיל; דפים קרים הם דפים "זקנים" (לפי הגדרת הפונקציה age), בעוד דפים חמים הם דפים "צעירים". הלוקליות הזו מתאימה במיוחד לאלגוריתם Generational אשר מצליח לאגד דפים עם גילאים דומים באותם הבלוקים ובכך לשפר את ביצועי הזיכרון.

חלון כתיבות

מודל ההנחות איתו עבדנו עד כה מניח כי סדרת הכתיבות ידועה לנו **במלואה.**

<u>תזכורת:</u> 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.

בפרקים הקודמים הראינו כיצד ניתן לתכנן אלגוריתמים אשר משתמשים בידע על הכתיבות העתידיות על מנת לשפר את ביצועי הזיכרון ולהוריד את ה-WA. למרות השיפורים היפים אשר הצלחנו להשיג באמצעות אלגוריתמים אלו, עלינו לזכור כי מודל ההנחות שלנו אינו פרקטי בחיים האמיתיים. במערכות אמיתיות לרוב לא נוכל לדעת ברגע עליית המערכת את עתיד הכתיבות המלא. מערכות המשתמשות בזיכרונות פלאש הן מערכות דינמיות אשר משתנות כל הזמן.

לדוגמה – זיכרון פלאש יכול להוות שכבת דיסק עבור מסד נתונים. מסד נתונים נמצא בשימוש תמידי על ידי משתמשים אשר מבצעים טרנזאקציות במערכת (הוספה/מחיקה/עריכה של רשומה). כל פעולה שמבצע משתמש צריכה "לחלחל" לבסוף אל הדיסק כדי שהשינויים שנעשו יהיו מגובים בדיסק (persistence). ברור שמערכת כזו לא תוכל לספק לנו את עתיד הכתיבות לדיסק מבעוד מועד.

למרות שנראה שמודל ההנחות הבסיסי שלנו אינו רלוונטי לעולם האמיתי, נרצה להציע מודל חלופי ומחמיר יותר אשר בו הפתרונות שהצענו יכולים להיות פרקטיים גם לשימושים האמיתיים של זיכרונות פלאש. נחזור לדוגמה של מסד הנתונים בה השתמשנו לעיל. מסדי נתונים לרוב אוכפים דרישות consistency. המשמעות הפשטנית של דרישה זו היא שלא נרצה לאפשר לשתי טרנזאקציות אשר רצות במקביל לבצע פעולות אשר עלולות להפר את השמורות אותן מסד הנתונים מתחייב לשמור. לדוגמה, נניח כי יש לנו מסד נתונים של בנק השומר פרטי חשבון של משתמשים שונים. מסד הנתונים מחויב לאכוף שאף משתמש לא נכנס למינוס בבנק. אם יש לנו חשבון עם 100 שקלים במאזן, לא נוכל לאפשר לשתי טרנזאקציות שמבצעות משיכה של 100 שקלים להסתיים בהצלחה. על מנת לאכוף דרישות אלו, מסד הנתונים שומר שינויים שמבצעת כל טרנזאקציה בזיכרון (הכוונה כאן בזיכרון היא לזיכרון RAM – volatile memory). כאשר המשתמש רוצה לסיים את הטרנזאקציה, מסד הנתונים מבצע תהליכי ולידציה על מנת לוודא שאין קונפליקטים, ורק אז כותב את המידע השמור בזיכרון לדיסק ומחזיר אישור למשתמש שהטרנזאקציה הסתיימה בהצלחה (commit). נשים לב, שבתהליך שתואר לעיל ישנו מידע רב שעלול להיכתב לזיכרון טרם הטרנזאקציה סיימה לרוץ, ואז תתבצע כתיבה מרוכזת של דפים אל הדיסק. ככל שהטרנזאקציה ארוכה יותר, כך עלולים להיות יותר דפים אשר נאגרים בזיכרון. שיטת האגירה של הדפים משתנה כתלות בייצוג הזיכרון, ויכולה להיעשות בעזרת שימוש ב-write buffers, write ahead logs או דרכים אחרות. המידע אשר נאגר בזיכרון עבור כל טרנזאקציה מכיל חלון של כתיבות אשר אנחנו יודעים שהולך להיכתב לדיסק. אם נסתכל על חלון הכתיבות הזה כעל רצף של דפים

המסודרים בצורה כרונולוגית, נקבל למעשה writing sequence עבורו העתיד ידוע לנו, וכעת נוכל להשתמש בכל האלגוריתמים והאופטימיזציות אשר הצגנו בפרקים הקודמים.

ננסח כעת את מודל ההנחות החדש שלנו:

בהינתן מספר הדפים לכתיבה N, נניח כי נתון לנו חלון של כתיבות בגודל n, כך ש-$0 \leq n \leq N$ המוגדר באופן הבא: $writing\_window = \{lp_1, lp_2, \cdots, lp_n\}$, כאשר $lp_i \in [0, U \cdot Z - 1]$ לכל $1 \leq i \leq n$.

מטרתנו בפרק זה תהיה להרחיב את היכולות של הסימולטור שלנו לתמוך בכתיבה באמצעות חלון כתיבות לפי המודל החדש שלנו. לאחר מכן נרצה לחקור את ביצועי האלגוריתמים שהצגנו בפרקים הקודמים עבור מודל הכתיבות החדש. נרצה להראות כי ככל ש-$n \to N$ האלגוריתמים מתכנסים לביצועי האלגוריתמים המקוריים שהוצגו עד כה, וככל $n \to 0$ האלגוריתמים מתכנסים לביצועי Greedy GC הסטנדרטי.

מימוש חלון הכתיבות

הרצת סימולציות עם תמיכה בחלון כתיבות התווספה לסימולטור. ניתן להריץ את כל אחד מהאלגוריתמים השונים תוך שימוש בחלון כתיבות משתנה. לשם פשטות, חלון הכתיבות של הסימולציה תמיד יהיה רציף ויקרה בתחילת ההרצה של הסימולציה. לדוגמה, בהינתן סימולציה עבור N כתיבות עם חלון כתיבות בגודל n, כתיבות הדפים בתחום $[0, n-1]$ יתבצעו על ידי אלגוריתם הכתיבה הנבחר (Generational/Greedy Lookahead) אשר מכיר את עתיד הכתיבות עד לכתיבת הדף ה-n-י. החל מהדף ה-$n+1$ ועד לדף ה-N הכתיבות יתבצעו בצורה רגילה תוך שימוש באלגוריתם Greedy GC. חלון הכתיבות יכול להיקבע להיות כל ערך בטווח $0 \leq n \leq N$. הבחירה בהפעלת חלון הכתיבות נעשית בעזרת פרמטר המועבר לסימולטור בזמן האתחול. במידה ונבחרה האפשרות להריץ את הסימולטור באמצעות חלון כתיבות, תופיע האפשרות לבחור את גודל החלון טרם התחלת הסימולציה.

בעת בחירה באלגוריתם Generational בשילוב חלון כתיבות, ישנו טיפול מיוחד בנקודת המעבר בין הכתיבה של הדפים תחת חלון הכתיבות לבין הכתיבה של יתרת הדפים. לאחר הכתיבה של הדף ה-n-י הבלוקים אשר מוגדרים כ-generation blocks מוכנסים אל ה-freelist, על מנת לאפשר לאלגוריתם Greedy GC להשתמש בהם כבלוקים רגילים לכתיבה של דפים.

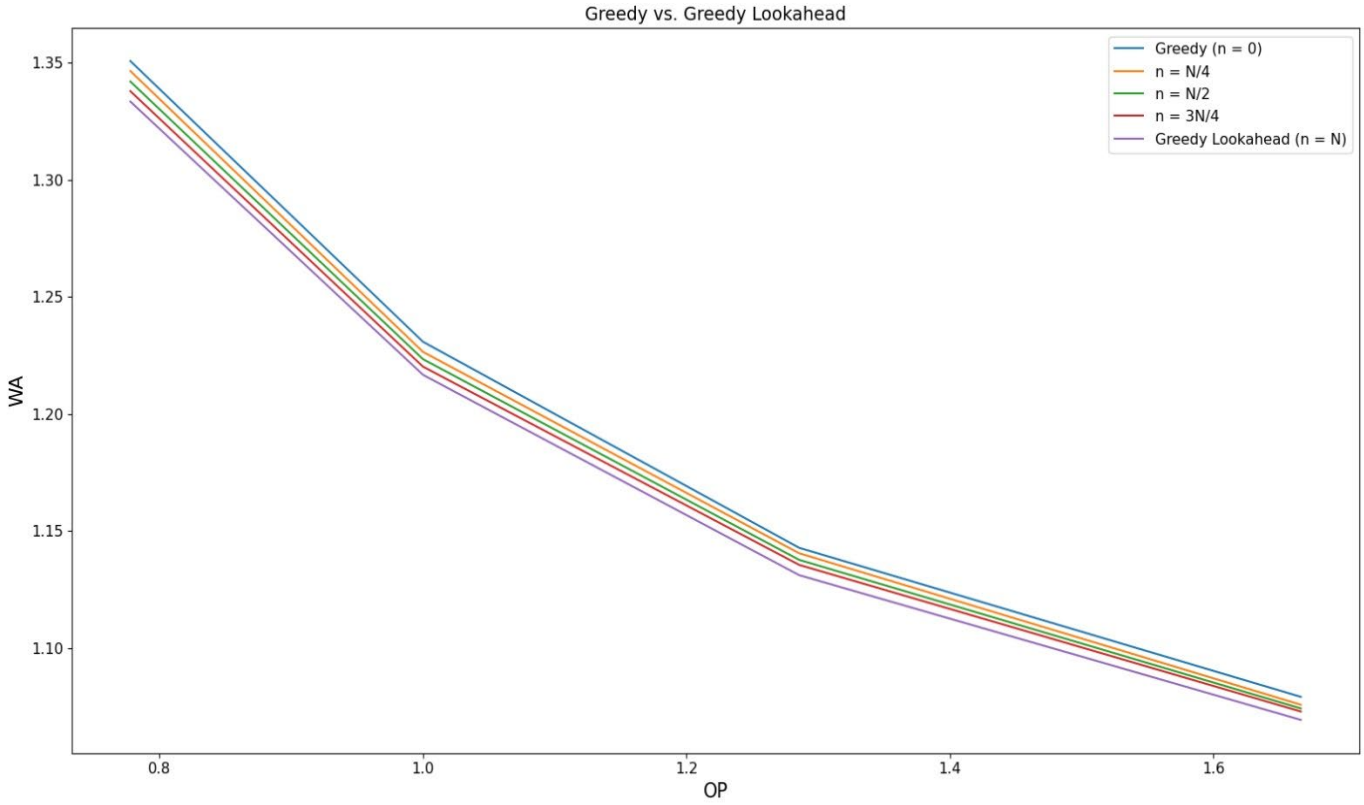
הוראות נוספות לעבודה עם חלון הכתיבות ניתן למצוא בקובץ README.md בדף ה-Github של הפרויקט.

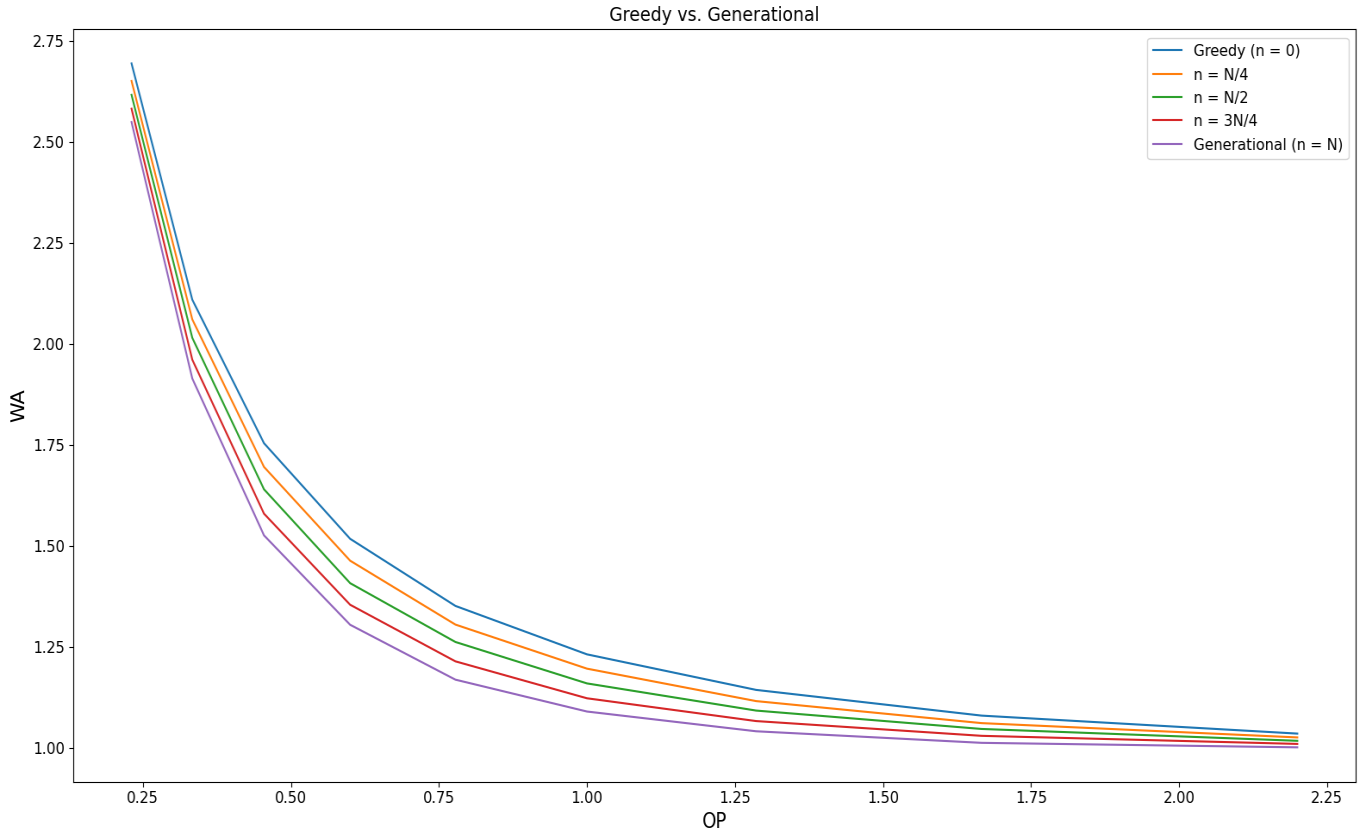
ניסויים

על מנת לאשש את ההשערות שלנו לגבי התנהגות הזיכרון בעת שימוש בחלון כתיבות, נערוך מס׳ ניסויים אשר יבחנו את התנהגות האלגוריתמים השונים בשילוב השימוש בחלון כתיבות.

ניסוי מס׳ 17 – ביצועי האלגוריתמים כתלות בגודל חלון הכתיבות

- תיאור הניסוי – נריץ את האלגוריתמים Greedy Lookahead ו-Generational בעזרת שימוש בחלון כתיבות משתנה. עבור כל אחד מהאלגוריתמים נבצע הרצות עבור גדלים משתנים של חלון כתיבות ונשווה בין הביצועים של האלגוריתם עבור כל גודל של חלון כתיבות.
- תנאי הניסוי – $T = 64, Z = 32, N = 10^5$. ערכי U נעים בטווח $12 \leq U \leq 60$ בקפיצות של 4. כל מדידה בניסוי היא של ערך write amplification. המדידה נקבעה על ידי לקיחת ממוצע של 10 הרצות עבור הפרמטרים הרלוונטיים. אלגוריתם Greedy Lookahead הורץ עם השיפורים אשר תוארו בפרק 2. עבור האלגוריתם Generational הניסוי הורץ עם מס׳ דורות קבוע אשר נקבע להיות $k = 2$, כאשר GC בוצע בעזרת אלגוריתם Greedy Lookahead עם השיפורים המתוארים בפרק 2. עבור כל אחד מהאלגוריתמים הניסוי הורץ עם 5 חלונות כתיבה שונים. חלונות הכתיבה בהם השתמשנו הם $n \in \left\{0, \frac{N}{4}, \frac{N}{2}, \frac{3N}{4}, N\right\}$. נשים לב כי עבור $n = 0$ אנחנו מקבלים את אלגוריתם Greedy GC המקורי, ועבור $n = N$ נקבל את האלגוריתם המקורי שלנו (Greedy Lookahead או Generational בהתאם לניסוי הרלוונטי).
- תוצאות הניסוי – נציג את תוצאות הניסוי בגרפים המתקבלים עבור כל אחד מהאלגוריתמים:


Greedy vs. Greedy Lookahead
Greedy (n = 0)
n = N/4
n = N/2
n = 3N/4
Greedy Lookahead (n = N)
WA
OP
1.35
1.30
1.25
1.20
1.15
1.10
0.8
1.0
1.2
1.4
1.6

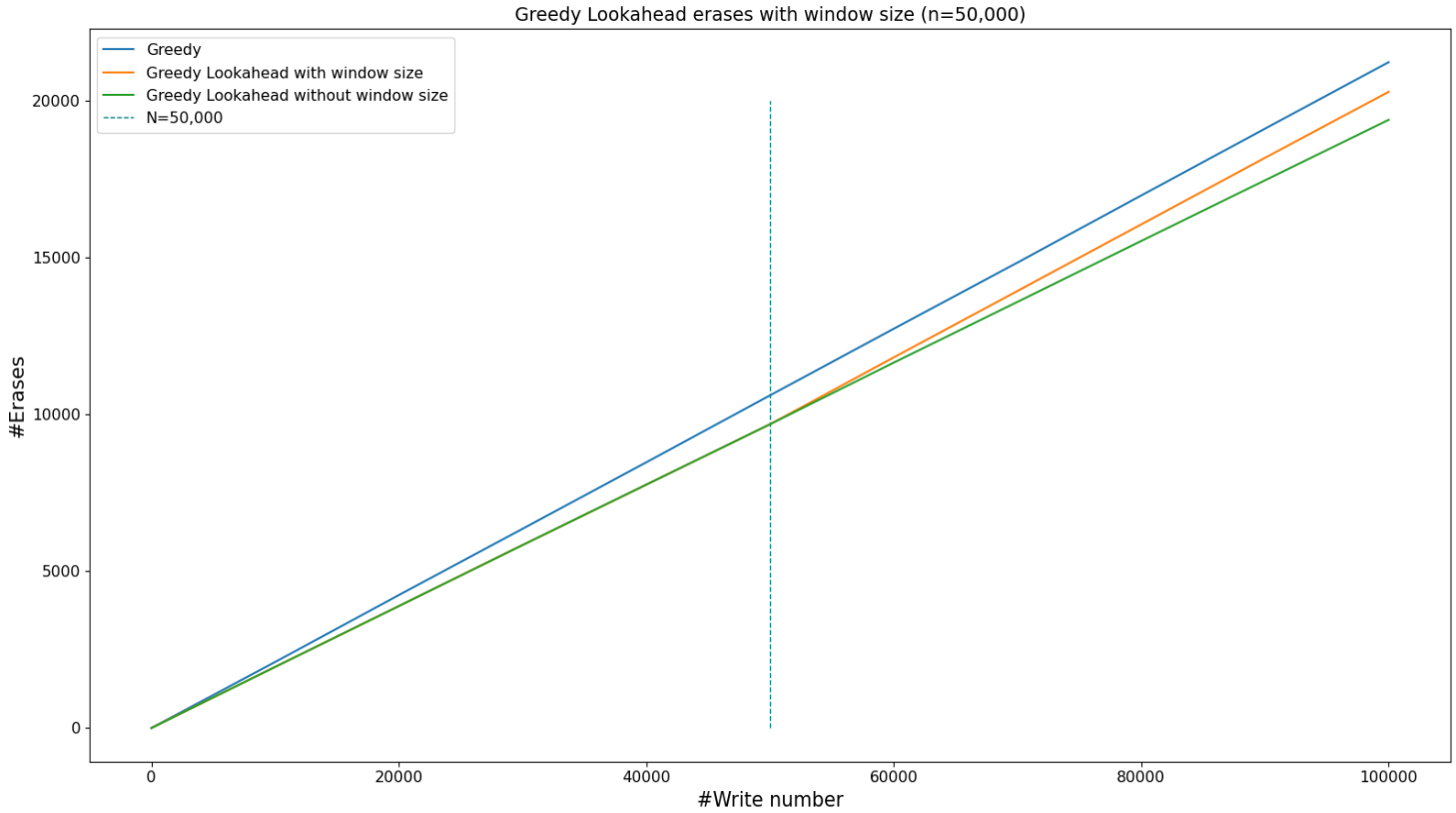

Greedy vs. Generational
Greedy (n = 0)
n = N/4
n = N/2
n = 3N/4
Generational (n = N)
WA
OP
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25

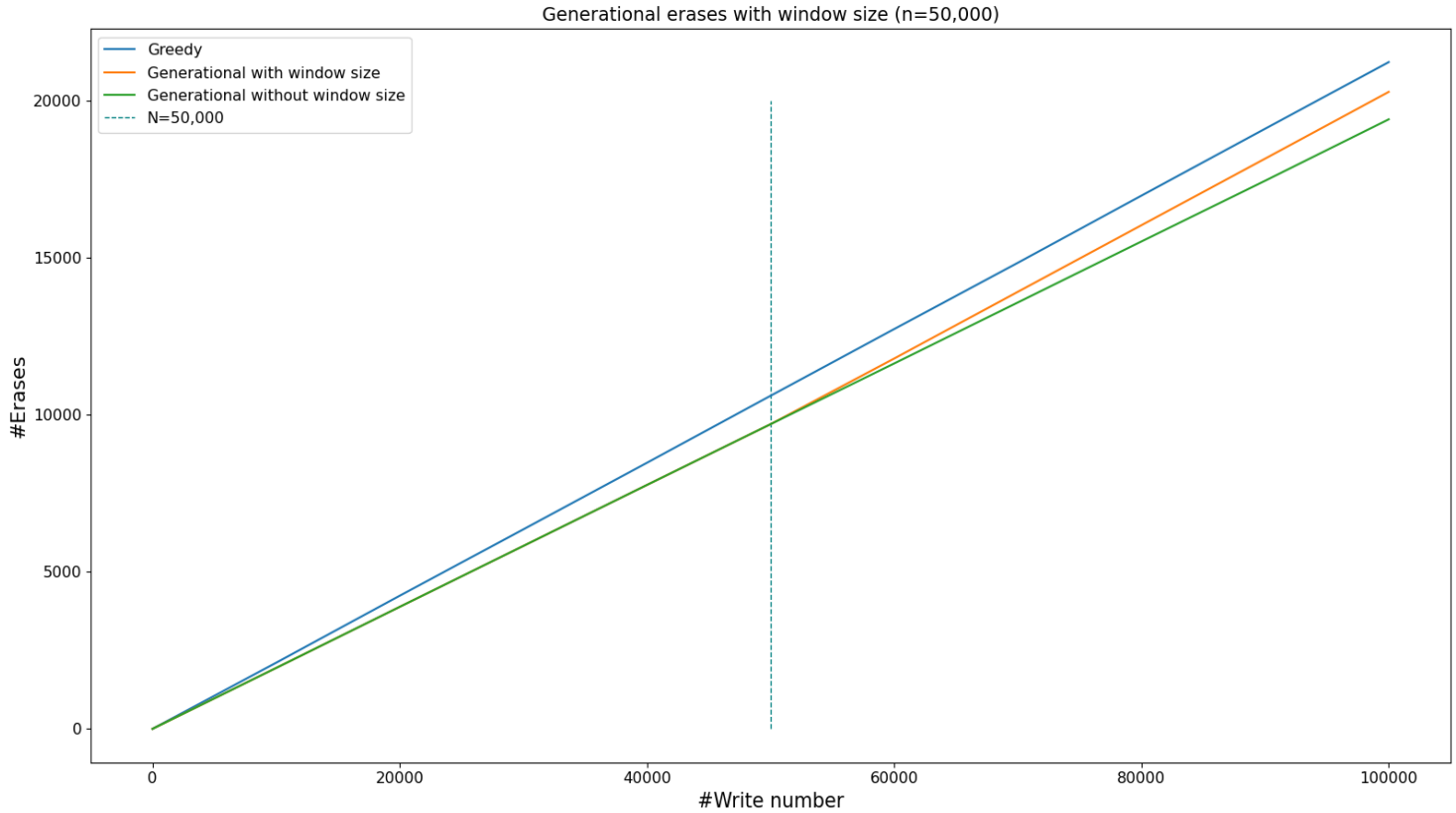
ניתן לראות כי עבור כל אחד מהאלגוריתמים אנחנו מקבלים את ההתנהגות אליה ציפינו – ככל שערכי $n \to N$ מתקבלים ביצועים טובים יותר השואפים לביצועי האלגוריתם המקורי (Greedy Lookahead או Generational), וככל ש-$n \to 0$ הביצועים פחות טובים ומתכנסים לביצועי אלגוריתם Greedy GC הסטנדרטי.

למרות שניסוי מס' 17 מראה בצורה יפה מאד את הבדלי הביצועים כתלות בבחירת חלון הכתיבות, הוא לא מצליח לבודד את ההשפעה של כל אחד ממקטעי הכתיבה על הביצועים של רצף הכתיבות כולו. נרצה להראות כי החלק בו אנחנו כותבים תחת חלון הכתיבות הוא החלק אשר משפר את הביצועים, בעוד החלק שאינו תחת חלון הכתיבות הוא החלק אשר "מקלקל" את הביצועים וגורם לנו להתכנס לכיוון ביצועי אלגוריתם Greedy GC.

ניסוי מס' 18 – השפעת מקטעי הכתיבות השונים

- תיאור הניסוי – נריץ את האלגוריתמים Greedy Lookahead ו-Generational בעזרת שימוש בחלון כתיבות בעל גודל **קבוע**. עבור כל אחד מהאלגוריתמים נמדוד את מספר מחיקות הבלוקים כתלות במספר הכתיבה בה אנחנו נמצאים. נרצה לבדוק כיצד משתנה דפוס המחיקות ברגע שאנחנו מסיימים את חלון הכתיבות ומתחילים לכתוב לפי אלגוריתם Greedy GC.
- תנאי הניסוי – $T = 64, U = 60, Z = 32, N = 10^5$. גודל חלון הכתיבות נקבע להיות $n = \frac{N}{2} = \frac{10^5}{2} = 50000$. אלגוריתם Greedy Lookahead הורץ עם השיפורים אשר תוארו בפרק 2. עבור האלגוריתם Generational הניסוי הורץ עם מס' דורות קבוע אשר נקבע להיות $k = 2$, כאשר GC בוצע בעזרת אלגוריתם Greedy Lookahead עם השיפורים המתוארים בפרק 2.
- תוצאות הניסוי – נציג את תוצאות הניסוי בגרפים המתקבלים עבור כל אחד מהאלגוריתמים. בכל תרשים מוצגים 3 גרפים – גרף אחד מתאר את המחיקות עבור סימולציה אשר הורצה אך ורק עם אלגוריתם Greedy GC, וסימולציה אשר הורצה אף ורק עם האלגוריתם Greedy Lookahead או Generational. הגרף השלישי הוא הגרף של הסימולציה שהורצה בעזרת חלון הכתיבות.


Greedy Lookahead erases with window size (n=50,000)
Greedy
Greedy Lookahead with window size
Greedy Lookahead without window size
N=50,000
#Erases
#Write number
0
5000
10000
15000
20000
0
20000
40000
60000
80000
100000


Generational erases with window size (n=50,000)
Greedy
Generational with window size
Generational without window size
N=50,000
#Erases
#Write number
0
5000
10000
15000
20000
0
20000
40000
60000
80000
100000

ניתן לראות כי עבור n הכתיבות הראשונות המבוצעות תחת חלון הכתיבות, הגרפים של האלגוריתם המקורי והסימולציה אשר משתמשת בחלון הכתיבות מתלכדים. החל מהכתיבה ה-$n+1$ מספר המחיקות עבור שני האלגוריתמים מתחיל לעלות ולהתקרב אל מס׳ המחיקות של אלגוריתם Greedy GC. בכך ניתן לראות בבירור את התרומה של כל אחד ממקטעי הכתיבה על התוצאה הסופית של הסימולציה.

סיכום

בפרק זה הראינו כיצד ניתן להתאים את מודל ההנחות שלנו ולהשתמש בחלון הכתיבות על מנת להפוך אותו למודל מעט יותר פרקטי ומתאים לעולם האמיתי. תחת המודל החדש הראינו כי עדיין ניתן להשתמש באלגוריתמים שפיתחנו בפרקים הקודמים כדי לקבל ביצועים אשר מביאים לשיפור בביצועים ביחס לאלגוריתם Greedy GC המקורי. הצלחנו להראות כי ישנו יחס ישר בין גודל חלון הכתיבות לבין ביצועי הסימולציות, ובנוסף הצלחנו לבודד את התרומות של כל אחד ממקטעי הכתיבה ולהראות כי הכתיבה תחת חלון הכתיבות אכן תורמת לשיפור הביצועים.

סיכום הפרויקט

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. גם לאחר הצגת האלגוריתמים הבסיסיים, נוכחנו שניתן להמשיך ולהעמיק בכל אחד מהאלגוריתמים השונים, להציע אופטימיזציות רבות ולבצע כיוונון לפרמטרים שונים אשר הביאו לשיפור ניכר בביצועים. האופטימיזציות התמקדנו גם בשיפור ביצועי הכתיבה וגם בשיפור סיבוכיות הזמן של האלגוריתמים.

על פניו, המודל והאלגוריתמים שהוצגו כאן הינם תאורטיים בלבד, משום שידע מקדים על כתיבות הוא מודל לא פרקטי עבור העולם האמיתי, אך בפרק האחרון הראינו כיצד ניתן לשנות במעט את הנחות המודל שלנו ולהשתמש בחלון כתיבות על מנת להפוך את המודל שלנו למודל אשר לו יישומים פרקטיים בהרבה מערכות אמיתיות. כמובן שעל מנת להשמיש את האלגוריתמים שהוצגו כאן (גם תחת מודל המשתמש בחלון כתיבות), יש לטפל בעוד לא מעט נושאים שמעט הזנחנו כאן, בראש ובראשונה בעיות סיבוכיות הזמן של האלגוריתמים. עם זאת, אנחנו סבורים כי ניתן להשתמש בחלק מהרעיונות והאלגוריתמים שהוצגו בפרויקט זה כבסיס לפיתוח כלים ורכיבי חומרה מהירים אשר יוכלו לשמש לייעל את הכתיבות לזיכרונות פלאש.

נספח א' – תוצאות ניסויים

- ניסוי מס' 10 – נציג את תוצאות הניסוי. השורה המכילה את ערכי ה-OF רלוונטית עבור הניסויים הבאים המוצגים בנספח.

OP / k	0.066	0.142	0.230	0.333	0.454	0.6	0.777	1	1.285	1.666	2.2	3	4.33
2	6.21305	3.59816	2.5571	1.91708	1.52521	1.30234	1.16868	1.08849	1.03998	1.0121	1.0008	1.00003	1
3	6.24239	3.59933	2.53899	1.94002	1.43835	1.22204	1.11555	1.05603	1.02182	1.00427	1.00024	1.00002	1
4	\	3.60781	2.52066	1.97291	1.38744	1.18803	1.09486	1.04435	1.01601	1.0025	1.00025	1.00004	1
5	\	3.61064	2.51106	1.99189	1.37017	1.17566	1.08749	1.03983	1.01347	1.00226	1.00028	1.00005	1
6	\	3.63658	2.51042	2.00542	1.37193	1.17193	1.08479	1.03832	1.01301	1.00206	1.00025	1.00006	1
7	\	3.65614	2.52388	2.02134	1.38792	1.17657	1.0861	1.03824	1.01289	1.00232	1.00029	1.00007	1
8	\	\	2.53322	2.0356	1.42274	1.1826	1.08788	1.03968	1.01404	1.00238	1.00031	1.00008	1
9	\	\	2.58393	2.05099	1.47094	1.19387	1.09262	1.04234	1.01465	1.00284	1.00034	1.00008	1
10	\	\	2.62131	2.06689	1.5418	1.21165	1.10003	1.04473	1.01653	1.00353	1.00041	1.00012	1
Best k	2	2	6	2	5	6	6	7	7	4	3	3	/
OF	24	18.66	13	48	11	8.88	8	8	8	6	6.66	10.66	/

- ניסוי מס' 11 – נציג את הטבלאות המתקבלות מהניסוי. הטבלה עבור $T = 64$ זהה לטבלה מניסוי מס' 10 המוצגת לעיל. השורה המכילה את ערכי ה-OF רלוונטית עבור הניסויים הבאים המוצגים בנספח. עבור $T = 64$ מוצגים רק ערכי k בטווח $2 \leq k \leq 10$.

תוצאות הניסוי עבור $T = 96$:

U / k	90	84	78	72	66	60	54	48	42	36	30	24	18
2	6.15355	3.55905	2.52792	1.89259	1.51347	1.29476	1.16527	1.08651	1.03782	1.01036	1.00054	1.00003	1
3	6.16019	3.54257	2.50919	1.89854	1.42139	1.21426	1.11085	1.0528	1.01957	1.00351	1.0003	1.00004	1
4	6.16107	3.54154	2.48382	1.93434	1.36572	1.17665	1.08861	1.04027	1.01295	1.00178	1.00031	1.00004	1
5	6.17106	3.5415	2.46948	1.94681	1.33201	1.15851	1.07913	1.03463	1.01123	1.00149	1.00032	1.00007	1
6	\	3.54627	2.46359	1.96248	1.31891	1.15074	1.07416	1.03167	1.00954	1.00134	1.00037	1.00007	1
7	\	3.55465	2.45425	1.97359	1.31683	1.14715	1.07218	1.03123	1.0095	1.00134	1.00038	1.00006	1
8	\	3.5702	2.46469	1.97917	1.3197	1.1481	1.07117	1.03036	1.0094	1.0014	1.00038	1.00008	1
9	\	3.58061	2.4669	1.98988	1.3343	1.15136	1.07255	1.03081	1.00985	1.00152	1.00044	1.00008	1
10	\	3.59568	2.4745	2.00435	1.34935	1.15536	1.07486	1.03153	1.01038	1.00162	1.00045	1.00008	1
11	\	3.60945	2.48812	2.01058	1.37812	1.16183	1.07569	1.0329	1.0108	1.00168	1.00042	1.0001	1
12	\	\	2.50096	2.02127	1.40815	1.16975	1.07976	1.03428	1.01116	1.00194	1.00053	1.0001	1
13	\	\	2.51693	2.03761	1.44585	1.18081	1.0838	1.03566	1.01217	1.0022	1.00051	1.0001	1
14	\	\	2.53989	2.0504	1.49634	1.1908	1.08816	1.03838	1.01253	1.00242	1.00054	1.00011	1
15	\	\	2.56484	2.06577	1.55279	1.20288	1.09296	1.04024	1.01404	1.00283	1.00064	1.00011	1
16	\	\	2.58531	2.07443	1.62533	1.21788	1.09873	1.04231	1.0149	1.00313	1.00064	1.00014	1
17	\	\	2.62004	2.08889	1.68267	1.23808	1.1072	1.04577	1.01605	1.00364	1.00066	1.00017	1
18	\	\	\	2.10307	1.73735	1.25501	1.11219	1.04962	1.01821	1.00398	1.00071	1.00016	1
19	\	\	\	2.11896	1.77026	1.27553	1.12464	1.05364	1.01984	1.0047	1.00074	1.00017	1
20	\	\	\	2.13116	1.79665	1.29656	1.13031	1.05713	1.02065	1.00512	1.00073	1.0002	1
Best k	2	5	7	2	7	7	8	8	8	6	3	2	/
OF	45	16.8	11.1429	36	9.42857	8.57143	6.75	6	5.25	6	10	12	/

תוצאות הניסוי עבור $T = 128$:

U / k	120	112	104	96	88	80	72	64	56	48	40	32	24
2	6.15059	3.54458	2.51561	1.87808	1.50598	1.29292	1.16343	1.08491	1.03718	1.00989	1.00053	1.00004	1
3	6.15165	3.5233	2.49855	1.87961	1.41402	1.21018	1.10746	1.05149	1.01861	1.00326	1.00038	1.00001	1
4	6.14777	3.5124	2.4691	1.90894	1.35501	1.17318	1.08662	1.03923	1.0125	1.0017	1.00037	1.00002	1
5	6.1453	3.50468	2.45094	1.929	1.32115	1.15282	1.07582	1.03307	1.01001	1.00152	1.00041	1.00004	1
6	6.15353	3.50232	2.44016	1.93823	1.29978	1.14242	1.06985	1.03034	1.0089	1.00146	1.00037	1.00004	1
7	6.15142	3.50257	2.43349	1.94515	1.28994	1.13834	1.06683	1.02864	1.00804	1.00145	1.00041	1.00006	1
8	\	3.51155	2.42687	1.95257	1.2885	1.13583	1.06487	1.02757	1.00829	1.00144	1.00044	1.00007	1
9	\	3.51798	2.42827	1.95563	1.29248	1.13481	1.06462	1.02777	1.00823	1.00158	1.00049	1.00007	1
10	\	3.52544	2.4341	1.96619	1.29748	1.13665	1.06533	1.02831	1.00847	1.00153	1.00045	1.00008	1
11	\	3.5432	2.43611	1.97153	1.30801	1.14087	1.06663	1.02797	1.00878	1.00168	1.00052	1.0001	1
12	\	3.54831	2.44342	1.98503	1.32286	1.14281	1.06758	1.02874	1.00907	1.00169	1.00057	1.00009	1
13	\	3.56238	2.45542	1.99461	1.33713	1.14822	1.06964	1.02926	1.0094	1.00174	1.00059	1.00012	1
14	\	3.5782	2.46417	2.00516	1.35695	1.15315	1.0724	1.03044	1.00962	1.0018	1.00059	1.00013	1
15	\	3.60491	2.47679	2.01351	1.38263	1.15906	1.07452	1.03154	1.0107	1.00172	1.00056	1.00012	1
16	\	\	2.48942	2.02538	1.4099	1.16509	1.07674	1.0331	1.01055	1.00201	1.00066	1.00013	1
17	\	\	2.5039	2.03333	1.44602	1.17507	1.08037	1.03393	1.01116	1.0021	1.00071	1.00014	1
18	\	\	2.52061	2.04453	1.47849	1.18371	1.08264	1.03617	1.01188	1.00228	1.00066	1.00017	1
19	\	\	2.53635	2.05685	1.5179	1.19373	1.08765	1.03782	1.01238	1.00253	1.00077	1.00015	1
20	\	\	2.56312	2.06608	1.56155	1.20165	1.09197	1.03912	1.01335	1.00284	1.0008	1.0002	1
Best k	5	6	8	2	8	9	9	8	7	8	6	3	/
OF	24	18.6667	13	48	11	8.88889	8	8	8	6	6.66667	10.6667	/

תוצאות הניסוי עבור $T = 160$:

U / k	150	140	130	120	110	100	90	80	70	60	50	40	30
2	6.1722	3.54151	2.51258	1.87153	1.50406	1.29073	1.16406	1.08566	1.03677	1.00987	1.00051	1.00004	1
3	6.17635	3.51878	2.49151	1.87209	1.4092	1.21043	1.10809	1.05137	1.01843	1.00317	1.0005	1.00002	1
4	6.15825	3.49959	2.46101	1.89163	1.35122	1.17006	1.08582	1.03827	1.012	1.00175	1.00046	1.00004	1
5	6.16517	3.48892	2.44096	1.9126	1.31379	1.1507	1.07386	1.03272	1.01013	1.00166	1.00045	1.00004	1
6	6.16372	3.4903	2.42818	1.92201	1.29166	1.14016	1.06775	1.02963	1.0087	1.00167	1.00045	1.00006	1
7	6.15771	3.48205	2.41756	1.92704	1.27954	1.1331	1.06473	1.02745	1.00812	1.00169	1.00049	1.00005	1
8	6.16685	3.48753	2.41123	1.9314	1.27295	1.12905	1.06236	1.02659	1.00768	1.00162	1.00045	1.00007	1
9	6.17758	3.48223	2.40701	1.93609	1.27184	1.12869	1.06163	1.02591	1.00797	1.00172	1.00051	1.00006	1
10	\	3.49111	2.40768	1.94094	1.27218	1.12742	1.0607	1.02564	1.00769	1.00175	1.00057	1.00008	1
11	\	3.49127	2.40752	1.94953	1.27689	1.12955	1.06177	1.02624	1.0078	1.00179	1.00058	1.00006	1
12	\	3.49644	2.41248	1.96001	1.2843	1.1307	1.06166	1.02638	1.00797	1.00195	1.00061	1.00012	1
13	\	3.51272	2.42231	1.96822	1.29442	1.13178	1.0631	1.02695	1.00816	1.00185	1.00058	1.00011	1
14	\	3.51999	2.42418	1.96966	1.30259	1.13692	1.06479	1.02759	1.00823	1.00193	1.00062	1.00012	1
15	\	3.53301	2.43802	1.98048	1.31664	1.13946	1.06526	1.02803	1.00873	1.00197	1.00067	1.00011	1
16	\	3.55523	2.44134	1.98996	1.3333	1.14276	1.06672	1.02885	1.00907	1.00206	1.0007	1.00013	1
17	\	3.56672	2.45514	1.9979	1.34886	1.14812	1.06947	1.02966	1.00957	1.00208	1.0007	1.00012	1
18	\	3.58318	2.46391	2.00815	1.36866	1.15448	1.07108	1.0302	1.00987	1.00211	1.00075	1.00015	1
19	\	3.60461	2.4794	2.01634	1.39044	1.15923	1.07353	1.03117	1.01032	1.00219	1.0008	1.00015	1
20	\	\	2.49098	2.02763	1.41411	1.16457	1.07585	1.03223	1.01051	1.00226	1.00078	1.00017	1
Best k	7	7	9	2	9	10	10	10	8	8	8	3	/
OF	21.4286	20	14.4444	60	12.2222	10	9	8	8.75	7.5	6.25	13.3333	/

- ניסוי מס' 12 – נציג את הטבלאות המתקבלות מהניסוי. הטבלה עבור $Z = 32$ ו-$T = 128$ זהה לטבלה שהצגנו בניסוי מס' 10, לכן נציג את שתי הטבלאות הנותרות עבור $Z = 64$ ו-$Z = 128$. השורה המכילה את ערכי ה-OF רלוונטית עבור הניסויים הבאים המוצגים בנספח. בעמוד הבא מוצגות תוצאות הניסוי עבור $Z = 64$, ובעמוד אחריו מוצגות תוצאות הניסוי עבור $Z = 128$.

U / k	120	112	104	96	88	80	72	64	56	48	40	32	24
2	7.01786	3.83951	2.6714	1.97514	1.56708	1.3342	1.19433	1.10581	1.05224	1.02057	1.00402	1.00025	1
3	7.03019	3.82896	2.64826	1.99799	1.47863	1.25107	1.13463	1.06968	1.03136	1.01045	1.00146	1.00025	1
4	7.0465	3.82465	2.62088	2.0277	1.42183	1.20879	1.11057	1.05563	1.02443	1.00705	1.00137	1.00023	1
5	7.07012	3.81745	2.60051	2.03689	1.38118	1.188	1.09916	1.04979	1.02099	1.00599	1.00139	1.00024	1
6	7.08903	3.82485	2.58927	2.04192	1.36198	1.17796	1.09256	1.0461	1.01956	1.00578	1.0013	1.00026	1
7	7.10952	3.83006	2.58944	2.04535	1.35194	1.17168	1.09042	1.04459	1.0183	1.0055	1.00145	1.00026	1
8	\	3.83963	2.58042	2.0493	1.34919	1.16938	1.08816	1.0438	1.01831	1.00545	1.00144	1.00028	1
9	\	3.85045	2.58549	2.05133	1.34848	1.16702	1.08752	1.04363	1.01801	1.0056	1.00148	1.00032	1
10	\	3.86005	2.59093	2.05856	1.36012	1.17129	1.08826	1.04344	1.0185	1.00555	1.00162	1.00033	1.00001
11	\	3.87371	2.59276	2.06881	1.37338	1.17369	1.08991	1.04416	1.01875	1.00578	1.00159	1.00036	1.00001
12	\	3.8952	2.60266	2.07202	1.39042	1.1787	1.09155	1.04495	1.01888	1.00597	1.00166	1.00036	1
13	\	3.89981	2.6114	2.08175	1.41074	1.18298	1.09335	1.04563	1.01936	1.00615	1.00171	1.00038	1
14	\	3.92737	2.62362	2.09007	1.44045	1.19055	1.09594	1.04676	1.02001	1.00657	1.00175	1.00042	1.00001
15	\	3.95369	2.6401	2.09923	1.47239	1.19731	1.09819	1.04795	1.02087	1.00676	1.00184	1.00045	1.00001
16	\	\	2.65919	2.10495	1.50629	1.20438	1.10087	1.0498	1.02117	1.00713	1.00193	1.00045	1.00002
17	\	\	2.67298	2.11912	1.55263	1.21368	1.10509	1.05154	1.02223	1.00709	1.00207	1.00052	1.00002
18	\	\	2.70376	2.12748	1.61883	1.22306	1.1094	1.05275	1.0228	1.00761	1.00202	1.0005	1.00005
19	\	\	2.72878	2.14005	1.68012	1.23396	1.11378	1.0559	1.02382	1.00788	1.00219	1.00054	1.00002
20	\	\	2.74364	2.14942	1.73154	1.24723	1.11911	1.05748	1.02499	1.00849	1.00222	1.00056	1.00005
Best k	2	5	8	2	9	9	9	10	9	8	6	4	2
OF	60	22.4	13	48	9.77778	8.88889	8	6.4	6.22222	6	6.66667	8	12

U / k	120	112	104	96	88	80	72	64	56	48	40	32	24
2	7.61909	4.01275	2.74999	2.02058	1.60281	1.36116	1.2153	1.12147	1.06406	1.02952	1.00928	1.00099	1.00004
3	7.6471	4.00771	2.73402	2.04815	1.5179	1.28044	1.15694	1.08577	1.04369	1.01874	1.00514	1.00082	1.00005
4	7.70752	4.00639	2.71078	2.06746	1.46273	1.23926	1.13333	1.07278	1.03628	1.01531	1.00426	1.0008	1.00005
5	7.73436	4.01008	2.69768	2.07747	1.42687	1.21775	1.12246	1.06653	1.03294	1.01359	1.0038	1.00095	1.00006
6	7.77752	4.01448	2.67982	2.07695	1.40194	1.20848	1.11668	1.06279	1.03126	1.01292	1.00379	1.00092	1.00006
7	7.83394	4.02664	2.68055	2.08109	1.39235	1.20279	1.11442	1.06234	1.03078	1.01248	1.0039	1.0009	1.00007
8	\	4.03827	2.6761	2.08013	1.38778	1.20294	1.11229	1.061	1.03028	1.01254	1.0039	1.00104	1.00008
9	\	4.0685	2.67775	2.08755	1.39295	1.20093	1.1123	1.06103	1.03069	1.01272	1.00401	1.00106	1.00011
10	\	4.07463	2.68696	2.09301	1.40554	1.20253	1.11301	1.06075	1.0307	1.01298	1.00413	1.00111	1.00012
11	\	4.09826	2.694	2.10291	1.412	1.20617	1.11379	1.06255	1.03093	1.01318	1.0043	1.00114	1.00012
12	\	4.11783	2.70495	2.11011	1.43303	1.20971	1.11605	1.06314	1.03126	1.01357	1.00445	1.00118	1.00013
13	\	4.14779	2.70978	2.11376	1.45875	1.21807	1.11962	1.06312	1.03186	1.01383	1.00469	1.00126	1.00015
14	\	4.15268	2.72124	2.12578	1.48032	1.22281	1.12087	1.06531	1.03272	1.01429	1.0048	1.0013	1.00015
15	\	4.18872	2.75431	2.13524	1.51751	1.23056	1.1234	1.067	1.03326	1.01449	1.00488	1.00129	1.00018
16	\	\	2.77504	2.14879	1.55537	1.23805	1.12591	1.06812	1.03426	1.01488	1.00506	1.00139	1.00018
17	\	\	2.79286	2.16211	1.60712	1.24782	1.13088	1.07064	1.03516	1.01522	1.00528	1.00144	1.00018
18	\	\	2.82015	2.16894	1.66564	1.25861	1.1354	1.07203	1.03653	1.01585	1.00537	1.00151	1.00019
19	\	\	2.84904	2.18117	1.72225	1.26997	1.14035	1.07462	1.03765	1.01674	1.00557	1.00163	1.00021
20	\	\	2.87073	2.19466	1.76334	1.28031	1.14657	1.0765	1.03858	1.01709	1.00583	1.00176	1.00023
Best *k*	2	4	8	2	8	9	8	10	8	7	6	4	2
OF	60	28	13	48	11	8.88889	9	6.4	7	6.85714	6.66667	8	12

- ניסוי מס' 13 – תוצאות הניסוי מוצגות כחלק מהטבלאות אשר הפקנו עבור ניסויים 10 ו-11. שתי השורות האחרונות בכל טבלה מציגות את מס' הדורות האופטימלי (עבור ערך ה-OP המתאים), ואת ערך ה-OF המתקבל.