פרויקט
תכנות מקבילי

אלרואי סעדיה: 315093419
דניאל זאדה: 318223278

תוכן עניינים

הקדמה

בעידן המודרני מערכות מחשוב מקביליות ורב ליבתיות הופכות נפוצות יותר ויותר , ולכן קיים צורך לשימוש במבני נתונים יעלים יותר לניהול התחרות בין החוטים הרבים שרצים בקביל (race).

חלק מהמבנים הנפוצים יותר הם תורים ותורי קדימויות, אשר משמשים לניהול משימות, תזמון תהליכים, ניתוב חבילות ברשתות, ומערכות מחשוב ענן.
תורים קונבנציונליים מבוססי נעילות יכולים לגרום למספר בעיות כגון: צוואר בקבוק, עיכובים במעבר נתונים, פגיעות ביעילות, תקלות טכניות ובעיות גמישות בין היתר.

על מנת לנסות לפתור בעיות אלו פותחו תורים ללא נעילות, שמאפשרים זרימה חלקה יותר של תהליכים ללא תלות בנעילה או בהגבלות שיכולות להוביל לעיכובים. תורים אלו מספקים פתרון גמיש ויעיל שמפחית את הסיכון לחסימות ומקל על הניהול של התור בצורה דינאמית ומותאמת למטרות שונות ומצבים משתנים.

בפרויקט זה נדבר על 2 סוגים שונים של תורי עדיפויות ללא נעילות:

1. CBPQ - תור קדימויות ללא נעילות שמשתמש ברשימות מקובצות (chunks) והוראת fetch and increment (F&I) להפחתת עומסים, ואלגורית elimination לשיפור ביצועים.
2. LPRQ – תור טבעתי ללא נעילות שמשפר את התור LCPQ על ידי ביטול הצורך בCAS2 מה שהופך אותו למודרני יותר ומותאם ליותר שפות שלא תומכות בפעולה זו (כמו למשל java ו-Kotlin).

השימוש בתורים מקביליים ללא נעילות מאפשר שיפור משמעותי בביצועים של מערכות בזמן אמת, במיוחד כאשר נדרש טיפול במספר רב של משימות במקביל. האלגוריתמים המוצגים בעבודה זו מציעים פתרונות שונים לניהול משאבים משותפים, וכל אחד מהם מצטיין בהיבטים שונים של הביצועים והסקלאביליות

CBPQ תוכנן להתמודד עם עומסים גבוהים תוך שמירה על יעילות מרבית של הפעולות Insert ו-DeleteMin, מה שהופך אותו לכלי חיוני במערכות הדורשות סדרי עדיפויות ברורים. לעומת זאת, LPRQ מציע גישה חדשנית המאפשרת ניהול טבעתי של התור ללא צורך בפעולות סנכרון כבדות, ובכך הוא מתאים לשפות ומערכות בהן אין תמיכה ב-CAS2.

בעבודה זו אנחנו נחקור את שני מבני הנתונים הלאה ועקרונותיהם, וננסה למצוא שיפור או הצעת ייעול על מנת לשלב את היתרונות של שני מבנים אלה למבנה אחד כך שהוא ייתן מענה לחסרונות של המבנים השונים.

שילוב של תורי CBPQ ו-LPRQ למבנה היברידי יכול להציע פתרון יעיל וגנרי שמתאים את עצמו באופן דינמי לעומסים שונים במערכת. בעומסים גבוהים, התור ההיברידי יבחר ב-CBPQ, שמספק ניהול סדרי עדיפויות טוב יותר ומפחית עומסים על ידי שימוש ברשימות מקובצות ואלגוריתם Elimination. כאשר העומס נמוך, המערכת תעבור לשימוש ב-LPRQ, שתור טבעתי ללא נעילות מספק פתרון קל יותר מבחינת משאבים, ומפשט את הסנכרון, במיוחד בסביבות בהן אין תמיכה ב-CAS2. השילוב הזה מאפשר למערכת להתמודד עם מגוון רחב של תרחישים, ובכך לשפר את הביצועים והסקלאביליות שלה על ידי התאמה אוטומטית של אלגוריתם התור לפי הצרכים המשתנים של המערכת.

CBPQ – תור עדיפויות יעיל ללא נעילות

CBPQ הוא תור עדיפויות ללא נעילות (Lock-Free), שנועד להתמודד עם צווארי בקבוק בתורים מסורתיים בסביבה מקבילית. בתורים מבוססי נעילות, מספר חוטים (threads) המנסים לגשת לתור בו-זמנית עלולים לגרום לעיכובים משמעותיים. CBPQ מציע גישה חדשה לניהול תורי עדיפויות, המאפשרת שיפור ביצועים במיוחד בתנאי עומס גבוה (high contention).

המבנה של CBPQ מבוסס על רשימה מקושרת של מקטעים הנקראים chunks, כאשר כל מקטע אחראי לטווח מסוים של ערכים (למשל ערכים בין 20-40 יכנסו לצ'אנק אחד). המקטעים מחוברים ביניהם במבנה של skip-list, שמאפשר גישה מהירה למקטע המתאים. המקטע הראשון מיועד למחיקות בלבד ומבצע את פעולת `DeleteMin`, בעוד המקטעים האחרים משמשים להכנסות (`Insert`). פעולות אלו מתחלקות בין המקטעים בהתאם לטווחי הערכים, מה שמקטין את הצורך בעדכונים מורכבים בתוך כל מקטע.

אחד היתרונות המרכזיים של CBPQ הוא השימוש בפקודת Fetch-and-Increment, שמאפשרת עדכון אטומי של אינדקסים ללא צורך במנגנוני סנכרון כבדים כמו CAS. גישה זו מבטיחה שתהליכים מרובים יוכלו לפעול במקביל מבלי להפריע אחד לשני. בנוסף, CBPQ עושה שימוש בטכניקת Elimination, שמפחיתה את העומס על מבנה הנתונים. בטכניקה זו, חוטים המבצעים פעולות הפוכות (כגון הוספה של ערך נמוך ומחיקת הערך המינימלי) יכולים "לבטל" אחד את השני, וכך להקטין את מספר הפעולות בפועל ולשפר את מהירות הביצוע.

המימוש של CBPQ משתמש במקטעים עם סטטוסים שונים: מקטע ראשון לטיפול במחיקות בלבד (סטטוס DELETE), מקטעים פנימיים לטיפול בהוספות (סטטוס INSERT), ובאפר (Buffer Chunk) לטיפול בערכים שצריכים להיכנס למקטע הראשון בעדכון הבא. כאשר מקטע מתמלא או מתרוקן, מתבצע תהליך Freezing, שבו המקטע מסומן כלא ניתן לשינוי, ולאחר מכן עובר עדכון על ידי פיצול או שילוב עם מקטעים אחרים. כך נמנעות התנגשויות בין תהליכים שונים המנסים לפעול על אותו מקטע.

אחד האתגרים המרכזיים של CBPQ הוא ניהול הזיכרון, שכן האלגוריתם אינו מבוסס על נעילות, ויש להבטיח שהזיכרון משתחרר בבטחה מבלי לגרום לקריסות תהליכים. לשם כך, האלגוריתם עושה שימוש בEpoch-Based Reclamation EBR, מנגנון שבו כל חוט מסמן את נקודת הזמן שבה הוא מתחיל להשתמש במקטע מסוים. כך ניתן לדעת מתי בטוח לשחרר את המקטעים הישנים שלא נמצאים עוד בשימוש.

מבחני הביצועים הראו כי CBPQ מצטיין במיוחד תחת עומסים גבוהים. כאשר הפעולות חולקו שווה בשווה בין הכנסות ומחיקות, CBPQ הציג שיפור של עד 80% בביצועים בהשוואה לאלגוריתמים אחרים. במצבים שבהם בוצעו רק מחיקות (רק פעולות של מחיקת המינימום) CBPQ היה מהיר פי 5 מהמתחרים בזכות השימוש היעיל בניהול זיכרון ובחלוקת העומס בין החוטים. עם זאת, כאשר העומס על המערכת נמוך, הכנסת ערכים קטנים עשויה לגרום לעיכובים, מכיוון שהערכים הללו צריכים להיכנס למקטע הראשון דרך מנגנון מיוחד שדורש עיבוד נוסף.

לסיכום, CBPQ מספק פתרון מתקדם ויעיל לניהול תורי עדיפויות מקביליים ללא נעילות, תוך שמירה על ביצועים גבוהים בתנאים של עומסים מרובים ומספר רב של תהליכים הפועלים במקביל. עם זאת, יש לשקול את השימוש בו בהתאם לדרישות המערכת ולתנאי העומס, במיוחד אם יש צורך בפתרון פשוט יותר כאשר הדרישה היא לניהול תורים בעומסים נמוכים.

LPRQ – תור מקבילי ללא 2CAS

תורים מקביליים משמשים לניהול משימות ותהליכים במערכות עתירות ביצועים. האלגוריתם LCRQ (Lazy Concurrent Ring Queue) מתבסס על באפר טבעתי (Ring Buffer) ופקודתFetch-and-Add לניהול הכנסות והוצאות בצורה אטומית. עם זאת, LCRQ מחייב שימוש בפקודת 2CAS (השוואה והחלפה של שני משתנים בו-זמנית), שאינה נתמכת בכל המעבדים והשפות. כתוצאה מכך, קיימת מגבלה על השימוש בו בפלטפורמות שונות.

המאמר מציע את LPRQ שזה Lock-Free Portable Ring Queue תור מקבילי שאינו דורש לבצע 2cas. האלגוריתם LPRQ משתמש בטוקן חוטים (Thread Token) ובמונה אפוקים(Epoch Counter)) כדי לתאם גישה לתאים בבאפר, תוך שימוש בפקודות אטומיות כמו (Compare-And-Swap (CAS ו-(Fetch-and-Add (FAA. שיטה זו מאפשרת ניהול הכנסה (Enqueue) והוצאה (Dequeue) בצורה מסונכרנת, מבלי לפגוע בסדר הנתונים.

האלגוריתם עובד באמצעות מבנה של טבעת (Ring Buffer), כאשר לכל תא בטבעת יש מזהה Epoch Counter, שמאפשר לוודא את סדר הגישה לתאים. כאשר תהליך רוצה להכניס נתון (Enqueue), הוא מבצע Fetch-and-Add על Tail, כדי לקבל מקום פנוי בטבעת, ואז מאמת באמצעות CAS שהתא באמת פנוי. כאשר תהליך רוצה למחוק נתון (Dequeue), הוא מבצע Fetch-and-Add על Head, מוודא שהתא מכיל נתון תקף, ואם כן – מחלץ את הערך. השימוש במונה האפוקים מאפשר להתמודד עם תהליכים שפועלים בו-זמנית מבלי להפעיל נעילות, כך שאין צורך לבדוק אם חוט אחר נמצא באותו שלב.

האלגוריתם מחליף את 2CAS בשלוש פעולות CAS נפרדות, אך עדיין מציג ביצועים דומים ל-LCRQ. תוצאות המאמר מראות כי האלגוריתם מהיר פי 1.6 מאלגוריתמים אחרים שאינם משתמשים ב-2CAS, ובעל יעילות גבוהה במיוחד כאשר מספר הצרכנים (Consumers) עולה על מספר המפיקים (Producers).

LPRQ פועל על עיקרון של מניעת התנגשויות בגישה לזיכרון, מה שמקטין את הצורך בתיאום מורכב בין חוטים. במקום להשתמש במנגנוני סנכרון מסורבלים, הוא מסתמך על קידום אטומי של אינדקסים בבאפר, כך שכל חוט משיג גישה בטוחה לתא המתאים מבלי להמתין לפעולות אחרות.

אחד האתגרים המרכזיים של LPRQ הוא התחרות על פעולות CAS, שעלולה לגרום לעומס במקרים של מספר רב של חוטים המנסים לגשת לאותו תא במקביל. עם זאת, האלגוריתם מתאים לשימוש רחב, במיוחד במערכות שבהן אין תמיכה ב-2CAS.

בסיכום, LPRQ מהווה פתרון נגיש יותר לניהול תורים מקביליים ללא נעילות, תוך שמירה על ביצועים גבוהים גם בסביבות בהן אין תמיכה בפקודות אטומיות מתקדמות. השימוש במונה אפוקים ובמנגנון טוקן חוטים מאפשר לו לתפקד ביעילות גם תחת עומסים גבוהים, ומציע חלופה ישימה ל-LCRQ במערכות שונות.

הפתרון שלנו

האידיאל מאחורי הרעיון היה לקחת את המבנה של CBPQ ולשפר את ניהול הצ'אנקים, כך שכל Chunk ינוהל בצורה יעילה יותר. ב-CBPQ המקורי הצ'אנקים נוהלו באמצעות מערך, דבר שדרש גישה ישירה לזיכרון וגרם להתנגשויות רבות בין חוטים, בגרסה החדשה השתמשנו במבנה טבעתי (Ring Buffer) המבוסס על LPRQ לניהול כל Chunk, מה שהפך אותו ליעיל במיוחד בעומסים נמוכים.
באמצעות שילוב הרעיונות משני המודלים, ניתן ליצור מבנה נתונים הממזג את היתרונות של כל אחד מהם, תוך פיצוי על החסרונות שלהם.

מצד אחד, CBPQ מצטיין בעומסים גבוהים, שכן הוא שומר על סנכרון יעיל ללא מנעולים (Lock-Free) ומאפשר ניהול סדר הנתונים בצורה מדויקת. עם זאת, החיסרון שלו הוא שבעומסים נמוכים נדרשות פעולות רבות להכנסת ושליפת איברים, מה שעלול ליצור תקורה מיותרת. מצד שני, LPRQ מבוסס על תור טבעתי ומשפר את LCRQ על ידי החלפת מנגנון 2CAS ב-CAS, מה שמקל על הסנכרון ומאפשר ביצועים טובים יותר בעומסים נמוכים, אך מציג ירידה משמעותית בביצועים בעומסים גבוהים.

בגרסה החדשה, אנו שואפים לשלב את היתרונות של שני המודלים, כך שנשמור על סנכרון ללא מנעולים, נקטין התנגשויות בין תהליכים, נשפר את היעילות בעומסים משתנים, ונבטיח שכל תהליך מסיים את פעולתו בצורה בטוחה וללא כניסה ללולאות אינסופיות

2CBPQ הוא גרסה משופרת של CBPQ שמשתמשת ב-LPRQ כדי להפוך את התור ליעיל יותר בסביבה מרובת חוטים. ה-CBPQ המקורי כבר היה Lock-Free, הוא פעל באמצעות רשימות מקושרות או מערכים והוביל לבעיות ביצועים בזמן שהרבה חוטים ניסו לבצע פעולות במקביל. ב-2CBPQ המשופר, כל Chunk בתור משתמש במבנה מעגלי (Ring Buffer), המאפשר הכנסה ושליפה אטומית ויעילה, עם מינימום התנגשויות בין חוטים. השילוב של CBPQ עם LPRQ ב-2CBPQ משפר משמעותית את מהירות פעולות התור, מקטין עיכובים, ומאפשר סקלאביליות גבוהה תחת עומסים כבדים.

בעוד שב-CBPQ המקורי כל Chunk נשמר כמערך בגודל קבוע מראש, ב-2CBPQ כל Chunk (למעט הראשון) משתמש ב-LPRQ, כלומר טבעת מעגלית עם גישה אטומית לערכים. שינוי זה מאיץ את ההכנסה והשליפה, שכן כל חוט יכול לקרוא ולכתוב ישירות לזיכרון פנוי ללא המתנה לנעילה. בנוסף, כאשר Chunk מתמלא, 2CBPQ יוצר Chunk חדש באופן דינמי, ללא צורך בהעתקת נתונים או השהיות מיותרות. כך, התור מתרחב בצורה חכמה, נמנעים צווארי בקבוק, והביצועים נותרים יציבים גם תחת עומסים גבוהים.

כאשר מספר חוטים מתחרים על אותו Chunk, עלולות להיווצר התנגשויות שיגרמו להאטה בביצועים. ב-2CBPQ, השימוש ב-CAS בתוך LPRQ מאפשר לכל חוט לגשת ישירות לזיכרון ולהכניס או לשלוף נתונים מבלי לחסום אחרים. לדוגמה, אם חוט מנסה להכניס ערך לתא שכבר נתפס, הוא פשוט מנסה מחדש, במקום להמתין לנעילה גלובלית. בנוסף, כאשר חוט אחד שולף ערך (()DeleteMin), שאר החוטים עדיין יכולים להכניס ולהוציא ערכים במקביל, ללא השפעה על הביצועים. ב-2CBPQ הביצועים נשארים יציבים גם תחת עומס כבד, מכיוון שאין צורך לעצור את כל התור כדי לבצע את הפיצול. כתוצאה מכך, המערכת ממשיכה לפעול במהירות וביעילות, גם כאשר יש כמות גדולה של חוטים עובדים בו-זמנית.

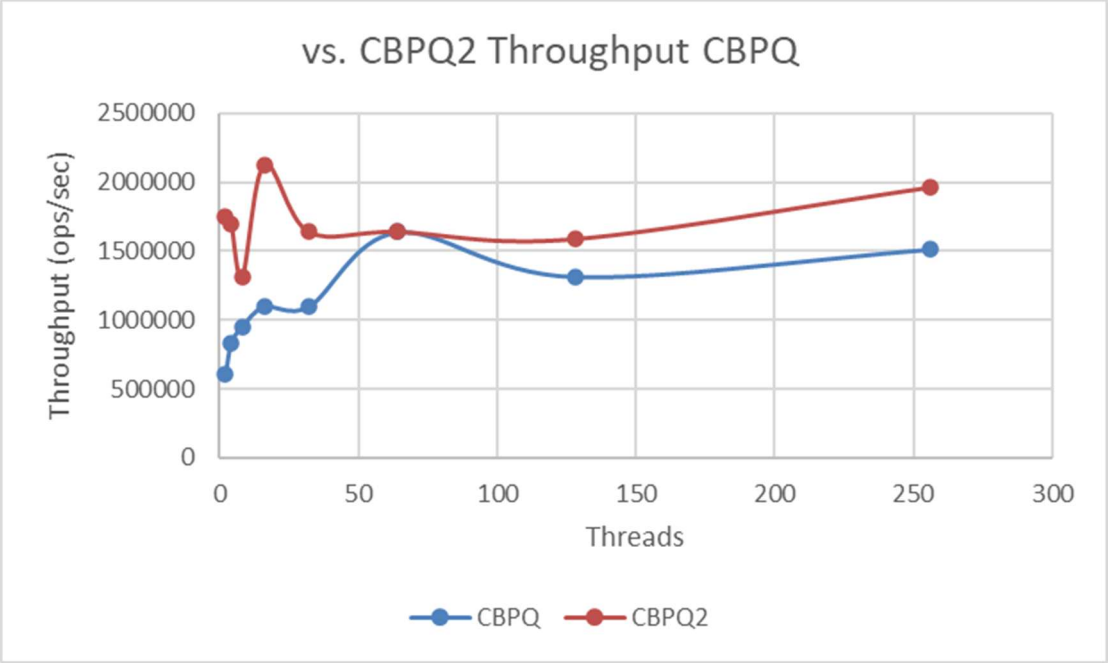
מדידות ביצועים מוכיחות את השיפור – זמן הריצה של 2CBPQ קצר יותר משמעותית, והוא מסוגל לבצע יותר פעולות לשנייה מאשר CBPQ, במיוחד תחת עומס כבד. כל התהליכים עובדים במקביל בלי לחכות אחד לשני, ולכן הביצועים נשארים מהירים ויציבים גם עם מעל 200 חוטים.

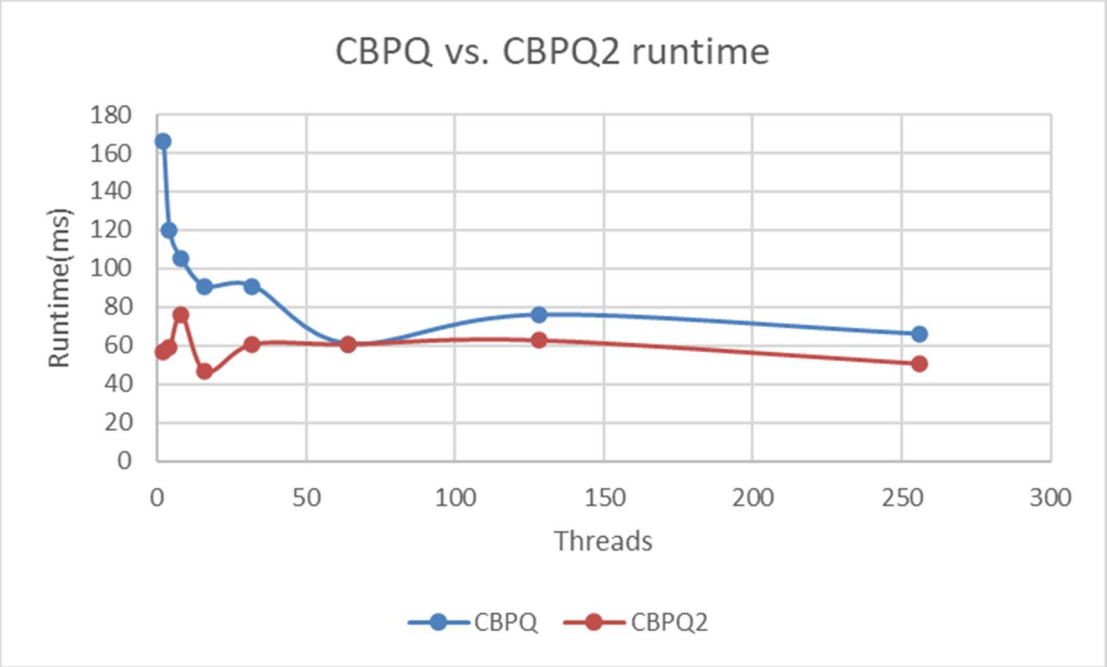
תוצאות

כעת נראה את התוצאות של הפרויקט- מימשנו את התור של CBPQ וCBPQ 2 במימוש שלנו שמתבסס על הקונספטים במאמר, וב-2CBPQ הוספנו את השיפור המוצע. לאחר מכן כתבנו קובץ שמריץ את 2 התורים בזה אחר זה עם אותם הפרמטרים לבדיקה. בקובץ אנחנו בוחרים בוליאני אקראי ומבצעים את ההוספה וההוצאה מהמבנה באופן אקראי לחלוטין כדי לדמות באמת מערכות מרובות חוטים שאין לנו שליטה על האופן בו ההוצאה וההכנסה מתבצעת, מה שמתקרב להדמיה של העולם האמיתי. בנוסף בדקנו עומסים שונים עם מספר שונה של חוטים: 2, 4, 8, 16, 32, 64, 128, 256.

להלן תוצאות הרצה שמשוות בין הביצועים של CBPQ ל2CBPQ, המדד שלנו זה כמות אופרציות שצריך לבצע ונמדוד את יעילות המבנה נתונים שלנו על ידי זמן לביצוע, וכמות אופרציות בשנייה כמו שלמדנו בכיתה. בהמשך נסיק מסקנות:

Threads	CBPQ Runtime (ms)	CBPQ Throughput (ops/sec)	CBPQ2 Runtime (ms)	CBPQ2 Throughput (ops/sec)
2	166	602409.64	57	1754385.96
4	120	833333.33	59	1694915.25
8	105	952380.95	76	1315789.47
16	91	1098901.1	47	2127659.57
32	91	1098901.1	61	1639344.26
64	61	1639344.26	61	1639344.26
128	76	1315789.47	63	1587301.59
256	66	1515151.52	51	1960784.31





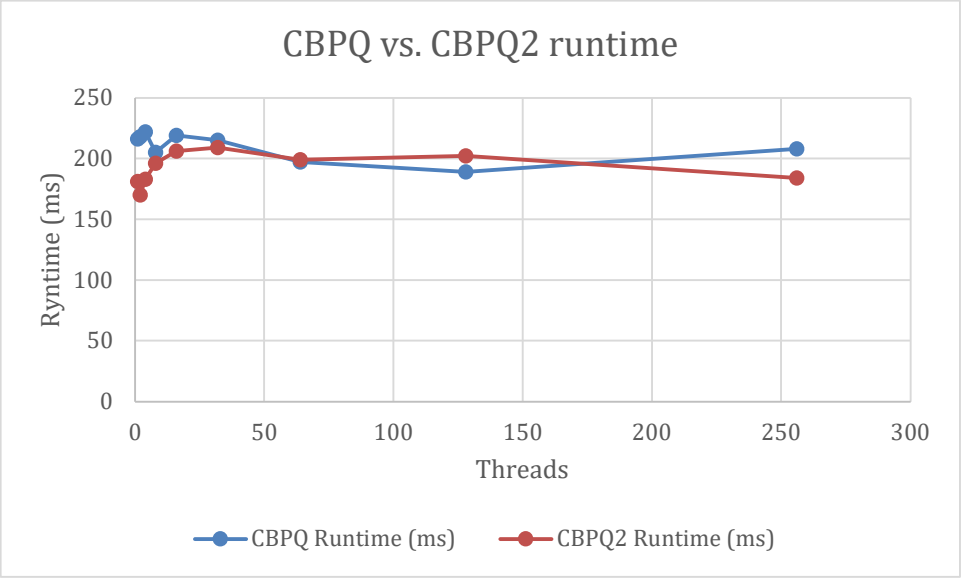
תוצאות ריצה לדוגמה(מהשרת) ל1000000 אופרציות:

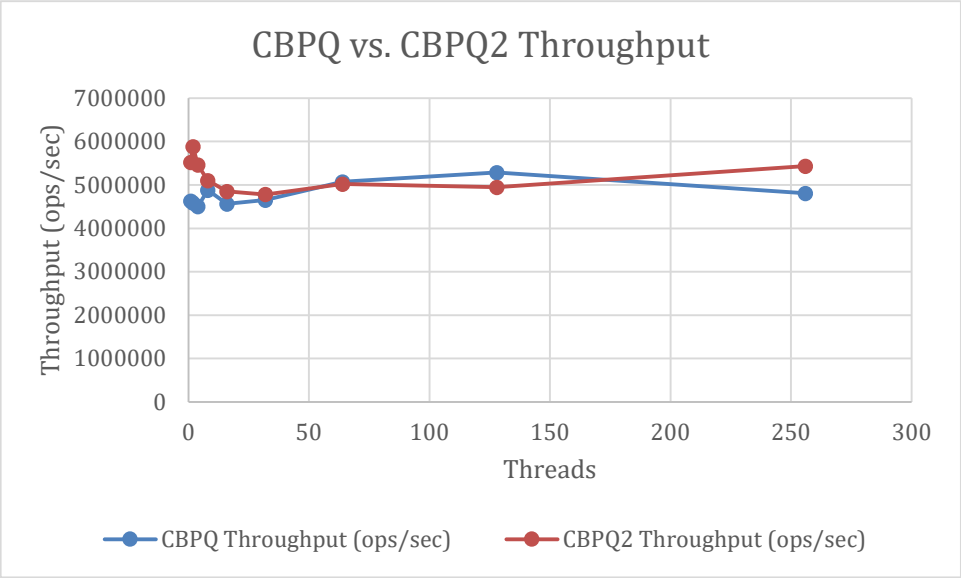
```
Results for: CBPQ
Threads     Runtime (ms)     Throughput (ops/sec)
2           51               1960784.31
4           43               2325581.40
8           33               3030303.03
16          25               4000000.00
32          30               3333333.33
64          26               3846153.85
128         33               3030303.03
256         45               2222222.22

Results for: CBPQ2
Threads     Runtime (ms)     Throughput (ops/sec)
2           23               4347826.09
4           22               4545454.55
8           23               4347826.09
16          27               3703703.70
32          25               4000000.00
64          28               3571428.57
128         23               4347826.09
256         47               2127659.57
```

תוצאות נוספות:

10 מיליון פעולות:





Results for: CBPQ

Threads	Runtime (ms)	Throughput (ops/sec)
1	216	4629629.63
2	218	4587155.96
4	222	4504504.50
8	205	4878048.78
16	219	4566210.05
32	215	4651162.79
64	197	5076142.13
128	189	5291005.29
256	208	4807692.31

Results for: CBPQ2

Threads	Runtime (ms)	Throughput (ops/sec)
1	181	5524861.88
2	170	5882352.94
4	183	5464480.87
8	196	5102040.82
16	206	4854368.93
32	209	4784689.00
64	199	5025125.63
128	202	4950495.05
256	184	5434782.61

מסקנות

מהתוצאות של הפרויקט ניתן להסיק במספר נקודות עיקריות:

גרסת CBPQ2 המשופרת מציגה ביצועים טובים יותר בהשוואה ל CBPQ-המקורי – זמן ריצה קצר יותר ומספר פעולות בשנייה גבוה יותר, במיוחד בעומסים כבדים עם מספר רב של חוטים.

השילוב בין היתרונות של CBPQ (ביצועים גבוהים תחת עומס) לבין היתרונות של LPRQ (יעילות ועבודה חלקה בסביבות עם עומסים נמוכים) יוצר פתרון היברידי שמתאים את עצמו לעומסים משתנים ומפחית התנגשות בין חוטים.

השימוש במבנה טבעתי (Ring Buffer) לניהול כל Chunk ב CBPQ2- מאפשר גישה אטומית ומהירה לזיכרון, מה שתורם ליציבות ומהירות הפעולות גם כאשר יש תחרות רבה בין החוטים.

לסיכום, התוצאות מדגישות כי השיפור ב CBPQ2-מוביל לפתרון סקלאבילי ויעיל יותר לניהול תורי עדיפויות במערכות מקבילות, כאשר הוא מצליח להתמודד בצורה מיטבית עם אתגרי תיאום וסנכרון בסביבה מרובת חוטים.

ביבליוגרפיה

1. מאמר **CBPQ**

Braginsky, A., Cohen, N., & Petrank, E. (2016). *CBPQ: High Performance Lock-Free Priority Queue*. Euro-Par 2016: Parallel Processing, 9902, 460-474. https://doi.org/10.1007/978-3-319-43659-3_33

2. מאמר **LCRQ**

Romanov, R., & Koval, N. (2023). *The State-of-the-Art LCRQ Concurrent Queue Algorithm Does NOT Require CAS2*. Proceedings of the 28th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '23), 14-20. https://doi.org/10.1145/3572848.3577485

3. ספר קורס:

Herlihy, M., & Shavit, N. (2008). *The art of multiprocessor programming.* Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-370591-4.X5000-6

4. GitHub Repository:

https://github.com/danielZada97/Concurrent-Programming-project