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 condition).

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

כדי לפתור בעיות סנכרון והעיכובים שנגרמים מהשימוש בנעילות, פותחו תורים ללא נעילות.תורים אלו מאפשרים לתהליכים לפעול בצורה רציפה, כלומר, כל תהליך יכול להמשיך לעבוד מבלי להמתין לקבלת נעילה, מה שמפחית עיכובים ומונע חסימות. בזכות זה, ניהול התור נעשה גמיש ויעיל, ומתאים בקלות למצבים ומטרות שונות.

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 ב2CAS מה שהופך אותו למודרני יותר ומותאם ליותר שפות שלא תומכות בפעולה זו (כמו למשל java ו-Kotlin).

*תור טבעתי- מבנה נתונים שבו הנתונים מאוחסנים במערך מעגלי, כך שאחרי האיבר האחרון חוזרים להתחלה. זה מאפשר הכנסה והוצאה יעילות בלי צורך להזיז נתונים בזיכרון .(יוסבר בפירוט בהמשך).

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-2CAS.

בעבודה זו אנחנו נחקור את שני מבני הנתונים האלה והעקרונות שלהם, 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 לשפר את היעילות על ידי ניצול היתרונות של כל אחד מהם כדי לצמצם את החסרונות. הרעיון הוא להוריד את העומס בתוך הגושים של CBPQ (פירוט בהמשך) בעזרת מנגנוני סנכרון של תור טבעתי, מה שיעזור להתמודד עם הבעיות שקשורות להוצאה ומחיקה של איברים בתוך כל גוש. היתרון המרכזי של CBPQ הוא העבודה עם גושים נפרדים במקום מערך או תור יחיד, מה שמאפשר לכל גוש לפעול באופן עצמאי בלי להפריע לפעולות שמתרחשות בגושים אחרים. (הסבר מפורט יותר יופיע בהמשך).

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).

האלגוריתם של CBPQ מבצע 3 פעולות עיקריות:

1. **הכנסה לתור (Insert) –** בעת הכנסה של איבר נבצע את הפעולות הבאות: תחילה נחפש את הגוש המתאים לפי טווח הערכים של כל גוש. נבצע את החיפוש על ידי skip-list שהוא יעיל יותר מlist רגיל. לאחר שמצאנו את הגוש הנכון אנחנו נכניס לכל גוש את האיבר על ידי שימוש ב-Fetch and increment, פעולה אטומית שמחזירה את האינדקס הנוכחי להכנסה ומעלה את הערך שלו ב-1. במידה והאיבר נוסף לגוש הראשון, והגוש הראשון מלא, נכניס אותו לבאפר זמני. זה נכון רק לגוש הראשון.
במידה ומכניסים לגוש מלא תתבצע פעולת עזר של freeze and split- פעולה שאומרת כרגע שהגוש נמצא במצב "מוקפא" ואין לבצע עליו פעולות, שבמהלכו הוא מתפצל לשני תתי גושים בגודל של הגוש המקורי, שכל אחד מהם אחראי על חצי מהטווח של הגוש המקורי (למשל אם הטווח של כל גוש תחילה הוא 20-40, לאחר הפיצול יהיו שני גושים זהי לגודל של הגוש המקורי, רק שכל אחד יכיל טווח שונה: 20-30, 30-40).

2. **הוצאה של האיבר הקטן ביותר בתור (DeleteMin)-** מוחקת ומחזירה את האיבר בעל הערך הנמוך ביותר מהתור, המתוחזק תמיד בגוש הראשון. הפעולה מבוצעת באמצעות Fetch and Increment שמבטיח מחיקה אטומית ללא התנגשויות בין תהליכים. אם הגוש מתרוקן, מופעל מנגנון Freeze & Merge, שמבצע שחזור של הגוש הראשון על ידי איחודו עם איברים מה buffer-ומהגוש הבא.

3. **פיצול ומיזוג (Split & Merge) -** כאשר גוש מתמלא, מתבצע Freeze & Split, אשר מחלק את הגוש לשני גושים קטנים יותר כדי לשמור על איזון בתור. לעומת זאת, אם הגוש הראשון מתרוקן, מופעל Freeze & Merge, המאחד אותו עם איברים מגושים נוספים כדי ליצור גוש ראשון חדש. תהליכים אלה מבטיחים שהמבנה יישאר יעיל וימנע גישה למיקומים ריקים.

פעולות אלו מתחלקות בין המקטעים בהתאם לטווחי הערכים, מה שמקטין את הצורך בעדכונים מורכבים בתוך כל מקטע.

CBPQ הוכח כליניארזבילי מכיוון שלכל פעולה יש נקודת זמן יחידה שבה היא מתבצעת, בעיקר בעת הקריאה לFetch and Increment, המבטיחה שהוספה או מחיקה מתרחשת בדיוק פעם אחת וללא התנגשויות. בנוסף, האלגוריתם חסר נעילות (Lock-Free), כלומר לפחות תהליך אחד תמיד מתקדם, מה שמונע Deadlock. השימוש במנגנוני Freeze & Split ו Freeze & Merge-מבטיח שמבנה הנתונים יישאר תקין גם לאחר גידול או שינוי בגושים. מנגנון Elimination משפר יעילות בכך שהוא מאפשר לזווג פעולות הוספה ומחיקה ישירות, ללא צורך בגישה למבנה הנתונים המרכזי.

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.

במאמר הושווה לאלגוריתמיים נפוצים אחרים, למשל לאלגוריתמים Adaptive PQ, mounds PQ וLinden & Jonnson PQ.

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 CAS2

האלגוריתם LCRQ (Lazy concurrent ring queue) מתבסס על באפר טבעתי (Ring Buffer) ופקודת Fetch-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 CAS2. האלגוריתם LPRQ משתמש בטוקן חוטים (Thread Token) ובמונה אפוקים (Epoch counter) כדי לתאם גישה לתאים בבאפר, תוך שימוש בפקודות אטומיות כמו CAS ו-F&I. שיטה זו מאפשרת ניהול הכנסה (Enqueue) והוצאה (Dequeue) בצורה מסונכרנת, מבלי לפגוע בסדר הנתונים.

מבנה הנתונים תומך בשלוש פעולות:

1. **הכנסה לתור (Enqueue)-** בעת הכנסת איבר חדש לתור, מתבצע Fetch-and-Add (FAA) על המונה Tail, אשר קובע את האינדקס הבא שבו האיבר יתווסף. לאחר מכן, האלגוריתם בודק אם התא הרלוונטי פנוי ושמחזור הזמן של הפעולה מתאים. אם התנאים מתקיימים, מתבצעת פקודת Compare-and-Swap (CAS) כדי להוסיף את האיבר לתא. אם התא כבר תפוס או שמחזור הזמן אינו תואם, ההוספה תיכשל, והפעולה תבוצע מחדש עד להצלחה. כאשר הטבעת הנוכחית מתמלאת, האלגוריתם יוצר טבעת חדשה, מקשר אותה לרשימה המקושרת של הטבעות, ומעביר את ההכנסות החדשות אליה.

2. **הוצאה מהתור (dequeue)-** כאשר יש צורך להוציא את האיבר בעל הקדימות הגבוהה ביותר, מתבצע Fetch-and-Add (FAA) על המונה Head, אשר קובע את האינדקס הבא למחיקה. האלגוריתם בודק אם התא שנבחר מכיל איבר תקף ואם מחזור הזמן שלו תואם לפעולה. אם כן, מתבצעת פקודת CAS, שמוחקת את האיבר ומחזירה את הערך שלו. אם התא ריק, או שהאיבר בתוכו שייך למחזור זמן אחר, האלגוריתם מבצע ניסיון חוזר עד להצלחה. במידה והטבעת כולה מתרוקנת, המצביע Head יעבור לטבעת הבאה ברשימה המקושרת, וכך מובטח שהתור ממשיך לפעול ללא עיכובים.

3. **ניהול הטבעת והקצאת טבעת חדשה כאשר הטבעת מתמלאת (Ring menegment)-** כאשר הטבעת הנוכחית מתמלאת, היא נסגרת לפעולות Enqueue, והאלגוריתם מקצה טבעת חדשה. טבעת זו מקושרת לרשימה המקושרת של הטבעות הקיימות, והתהליכים שמבצעים הכנסה ממשיכים ישירות לטבעת החדשה. מנגד, כאשר הטבעת הראשונה בתור מתרוקנת לחלוטין, המצביע Head מועבר לטבעת הבאה, וכך התור נשאר קומפקטי ויעיל. תהליך זה מבטיח שהאלגוריתם מאוזן מבחינת שימוש בזיכרון, מונע התנגשויות בין תהליכים, ושומר על ביצועים אופטימליים במקביליות גבוהה.

האלגוריתם LPRQ פועל כך שכל פעולת הוספה או מחיקה מתבצעת באופן מסודר ובנקודה אחת בזמן, ללא שגיאות בגישה לנתונים. הוא משתמש בפעולות אטומיות כמו Compare-and-Swap (CAS)ו Fetch-and-Increment (F&I)- כדי למנוע התנגשויות בין תהליכים במקביל. בנוסף, הוא כולל ספירת מחזורים (Epochs) ומנגנון טוקן חוטים, שמבטיחים שכל חוט מבצע את הפעולה שלו בתור בצורה תקינה וללא הפרעה לאחרים.

LPRQ מהווה חלופה ל CAS2,- על ידי ביצוע שלוש פעולות CAS נפרדות במקום אחת, ועדיין מצליח לשמור על ביצועים דומים ל LCRQ.- תוצאות המאמר מצביעות על כך שהוא מהיר עד פי 1.6 מאלגוריתמים אחרים שאינם משתמשים ב CAS2,- במיוחד במצבים בהם מספר הצרכנים (Consumers) גבוה ממספר המפיקים (Producers). האלגוריתם נמנע ממנגנוני סנכרון כבדים בכך שהוא מנהל את האינדקסים בבאפר בצורה אטומית, כך שכל חוט משיג גישה בטוחה ללא צורך בהמתנה.

למרות שהתחרות על פעולות CAS עלולה לגרום לעומס כאשר חוטים רבים מנסים לגשת לאותו תא LPRQ מותאם במיוחד למערכות ללא תמיכה ב CAS2,- ומספק פתרון יעיל וחסר נעילות לניהול תורים מקביליים. שילוב מנגנון מונה אפוקים יחד עם טוקן חוטים מאפשר לו לתפקד בצורה יציבה ויעילה גם תחת עומסים גבוהים, מה שהופך אותו לבחירה מצוינת עבור מערכות עם ריבוי תהליכים

הפתרון שלנו

הרעיון שלנו הוא לשלב CBPQ עם תור טבעתי בתוך כל גוש, כך שנוכל לנהל כל גוש בנפרד בצורה יעילה ולמנוע פעולות מסובכות כמו הקפאה, פיצול או מיזוג, שפוגעות בביצועים. במקום שהשינויים ישפיעו על כל המבנה ויגרמו לעיכובים, השימוש בתור טבעתי בתוך כל גוש יאפשר עבודה רציפה וללא התנגשויות בין תהליכים.

הגושים ימשיכו להיות מנוהלים על ידי skip-list, וכל אחד מהם יכיל תור טבעתי כמו ב LPRQ,- תוך שימוש ב CAS-ו F&I- כדי לנהל את ההכנסה וההוצאה בצורה אטומית ויעילה. כך, כל גוש יפעל באופן עצמאי, מה שיקטין את החיכוך בין התהליכים, ישפר את ניהול הנתונים, וישמור על סדר הפעולות בצורה ליניארית.

בנוסף, הגוש הראשון לא יהיה שונה משאר הגושים, כי נוכל להפוך את הבאפר עצמו לתור טבעתי. זה יפשט משמעותית את מחיקת המינימום, כי כל הערכים יהיו בתוך התור עצמו, בלי הפרדה מלאכותית בין הבאפר לבין הגוש הראשון. כך נמנע מצב שבו הערך הקטן ביותר "נתקע" בבאפר במקום להיות נגיש ישירות, מה שיגרום למחיקות להיות מהירות יותר ויעילות יותר בלי האטות מיותרות.

נסתכל למשל על הדוגמה הבאה: ב-CBPQ הרגיל בעצם אנחנו יכולים להכניס לגוש הראשון (נניח שהגושים בגודל 10) את המספרים מ-2 עד 11 כולל, סה"כ 10 מספרים ונמלא את הגוש, לאחר מכן נכניס את המספר 1 לגוש. לפי האלגוריתם הרגיל של CBPQ ה1 ילך לבאפר ויחכה שם עד שנבצע בניה מחדש לגוש הראשון(אחרי שהוא מתרוקן לגמרי). במצב כזה יהיה לנו בתור את המספרים 1 עד 11 כאשר 1 נמצא בבאפר והשאר בגוש, לכן אם נבצע מחיקה של המינימום אנחנו/ נמחק את המספר 2 ולא את 1 מכיוון ש1 נמצא בבאפר ולא בתור. זה יקרה עד שנוציא את כל המספרים עד 11 ורק אז נכניס את הבאפר עם 1 לתוך הגוש הראשון. זאת אומרת שלא נוציא בהכרח את המספר המינימלי בתור בצורה כזאת. לעומת זאת בשיפור שאנחנו מציעים אנחנו נוציא תמיד את המינימום מהגוש הראשון מכיוון שנסיר את התלות בבאפר על ידי יצרית התור הטבעתי.

באמצעות שילוב הרעיונות משני המודלים, ניתן ליצור מבנה נתונים הממזג את היתרונות של כל אחד מהם, תוך פיצוי על החסרונות שלהם.

בעומסים נמוכים CBPQ ,סובל מתקורה משמעותית ,מכיוון שהוא מצריך ניהול גושים, חיפוש ב Skip-List, ופיצולים או מיזוגים ,גם כאשר מספר האיברים קטן ואין עומס כבד. לעומת זאת ,LPRQ , שמבוסס על תור טבעתי ,פועל ביעילות רבה יותר מכיוון שכל פעולה מתבצעת ישירות על הטבעת באמצעות CAS ו-F&I, ללא צורך במבני נתונים מורכבים.

מצד אחד CBPQ ,מצטיין בעומסים גבוהים בזכות סנכרון ללא מנעולים (Lock-Free) וניהול מסודר של סדר הנתונים ,אך בעומסים נמוכים הוא סובל מתקורה גבוהה עקב ריבוי פעולות מיותרות .מצד שני , LPRQ קל יותר לניהול בעומסים נמוכים מכיוון שהוא מתבסס על תור טבעתי פשוט ,אך חווה ירידה בביצועים בעומסים גבוהים בשל התנגשויות מוגברות בגישה לתאים.

בגרסה החדשה, אנו משלבים את היתרונות של שני האלגוריתמים :שמירה על סנכרון ללא מנעולים , הפחתת התנגשויות בין תהליכים ,שיפור ביצועים בעומסים משתנים ,והבטחה שכל תהליך יסיים את פעולתו ביעילות וללא לולאות אינסופיות. בנוסף אנחנו נוריד את הצורך בבאפר לגוש הראשון ונוריד את המיזוג או פיצול של הגושים דבר שימנע גם הקפאה של גושים בתהליכים אלו.

כעת נציג את השינויים המרכזיים באלגוריתם CBPQ ונראה איך הם משפרים את CBPQ2

על מנת לממש את הפתרון, נבנה מבנה נתונים חדש ונקרא לו 2CBPQ (כלומר גרסה חדשה של CBPQ), בתוכו נממש את הפעולות של הנכסה (insert) ומחיקת מינימום (DeleteMin) ובעצם המיזוג או הפיצול יהפכו למיותרים ולא נשתמש בהם, בנוסף בכל גוש במקום לשמור רשימה רגילה אנחנו בעצם נשמור תור טבעתי כמו שהגדרנו מקודם. המימוש הוא lock-free וגם משתמש בפעולות של CAS ו-F&I ושומר על ליניארזביליות עם תקורה יותר נמוכה מאשר בCBPQ הרגיל.

CBPQ2 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-CBPQ2 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-CBPQ2 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, CBPQ2 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-CBPQ2, 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-CBPQ2 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.

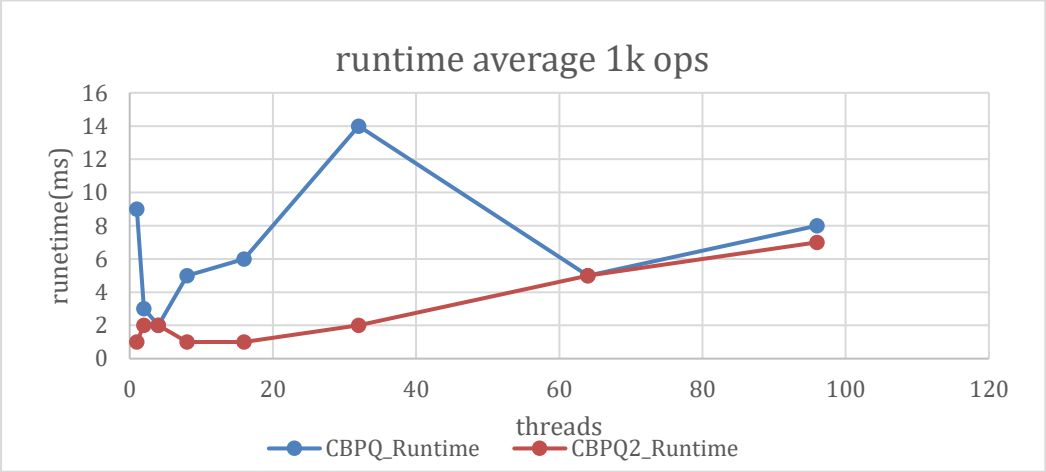
כדי לבדוק את הביצועים של CBPQ2 לעומת CBPQ בעומסים שונים, נבצע ניסוי עם עומסים נמוכים וגם עומסים גבוהים כלומר בין 1-96 חוטים ונבצע בין 1,000-1,000,000 פעולות. .בעומסים נמוכים נמדוד זמן ריצה ממוצע ותפוקה ,כדי לבדוק אם CBPQ2 מציג יתרון בהפחתת התקורה של ניהול הגושים. אנו מצפים ש CBPQ2-יראה יתרון משמעותי תחת עומסים גבוהים ,בגלל שהסרת מנגנוני פיצול ומיזוג גושים והתור הטבעתי בכל Chunk יפחיתו את ההתנגשויות וישפרו את הביצועים.

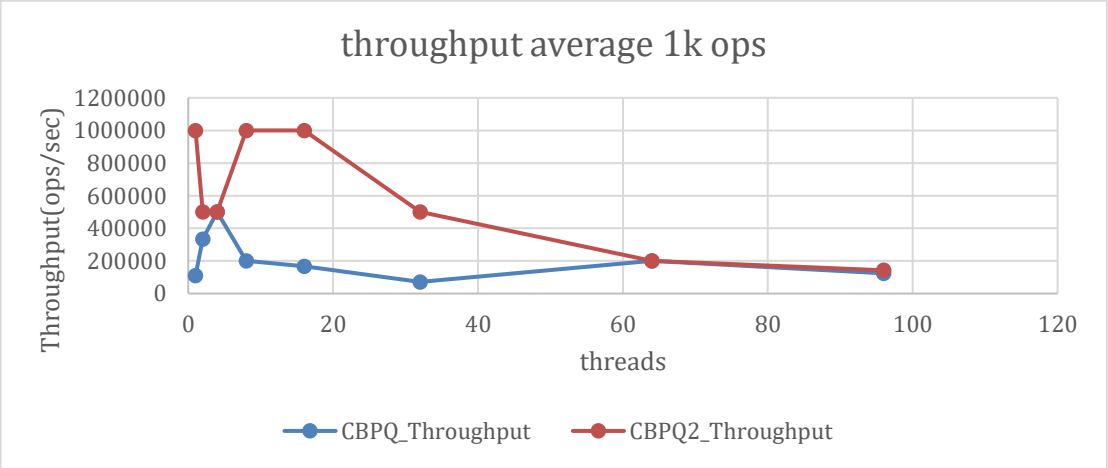
תוצאות

מימשנו מבני הנתונים CBPQ ו-CBPQ2 במימוש שלנו לפי הקונספט של המאמר, וב-CBPQ2 הוספנו את השיפור המוצע. כדי להשוות את הביצועים הרצנו על גבי השרת את שני המבנים עם אותם פרמטרים כדי שהמבחן יהיה כמה שיותר הוגן. במהלך ההרצה, כל חוט מבצע 90% פעולות הכנסה ו- 10% הוצאה. בנוסף בדקנו את ביצועי המימושים עם עומסים שונים ומספר חוטים שונה: (1, 2, 4, 8, 16, 32, 64 ו96) מספר האופרציות לכל ניסוי: (1000, 10000, 100000, 1000000) כדי להבטיח כמה שיותר דיוק ועומסים משתנים. כפי שציינו למעלה כל ניסוי הרצנו בשרת, 10 פעמים ואז חישבנו את הממוצע של זמני הריצה והביצועים שהתקבלו.

אלף הוספות\הוצאות:

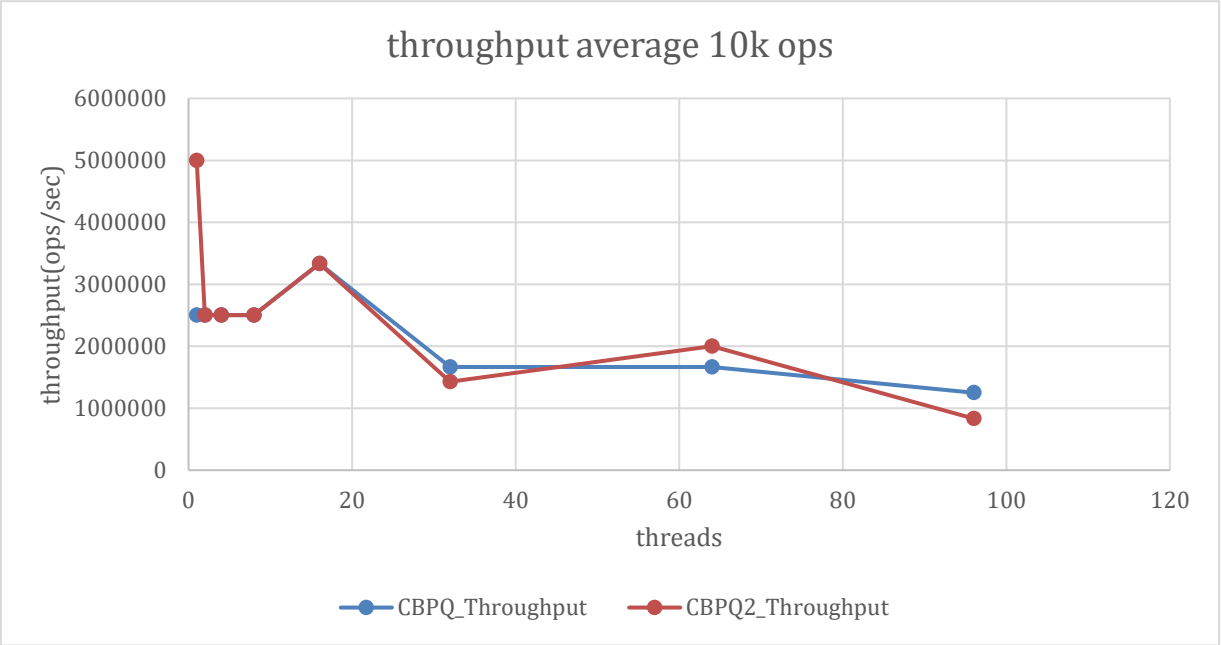
Threads	CBPQ_Runtime	CBPQ_Throughput	CBPQ2_Runtime	CBPQ2_Throughput
1	9	111111.11	1	1000000
2	3	333333.33	2	500000
4	2	500000	2	500000
8	5	200000	1	1000000
16	6	166666.67	1	1000000
32	14	71428.57	2	500000
64	5	200000	5	200000
96	8	125000	7	142857.14

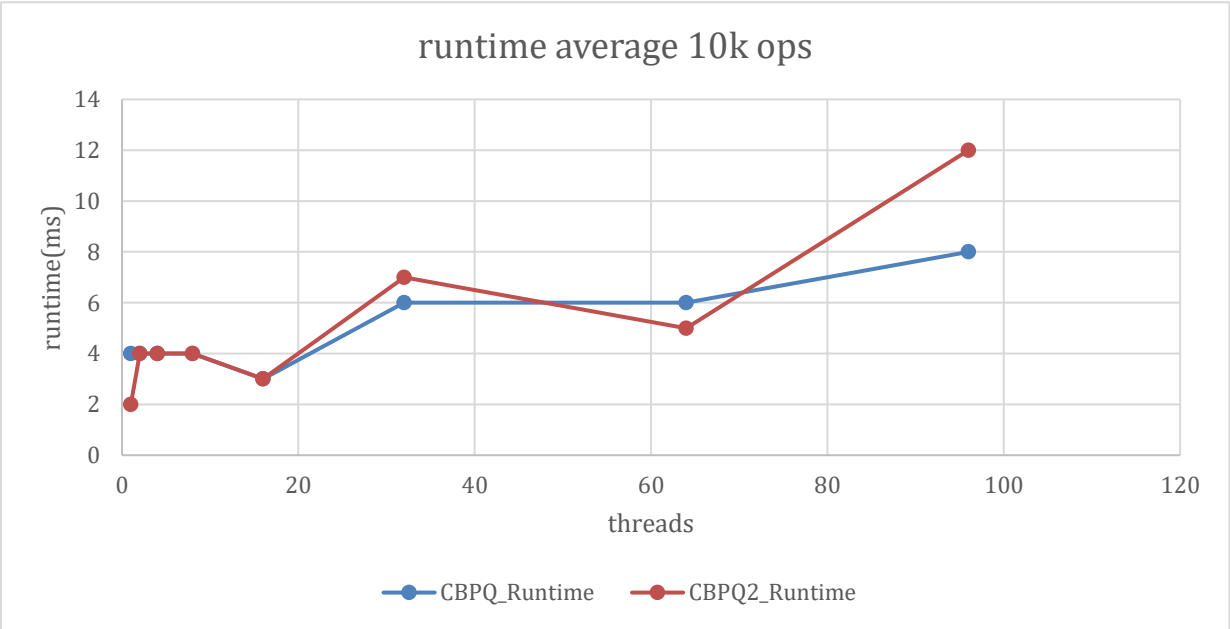




10 אלפים הוצאות\הכנסות:

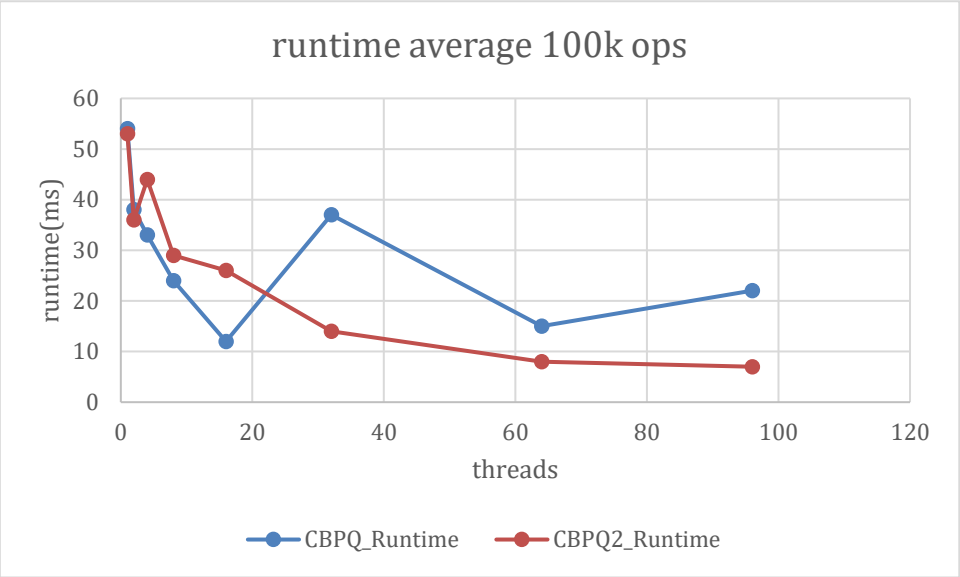
Threads	CBPQ_Runtime	CBPQ_Throughput	CBPQ2_Runtime	CBPQ2_Throughput
1	4	2500000	2	5000000
2	4	2500000	4	2500000
4	4	2500000	4	2500000
8	4	2500000	4	2500000
16	3	3333333.33	3	3333333.33
32	6	1666666.67	7	1428571.43
64	6	1666666.67	5	2000000
96	8	1250000	12	833333.33

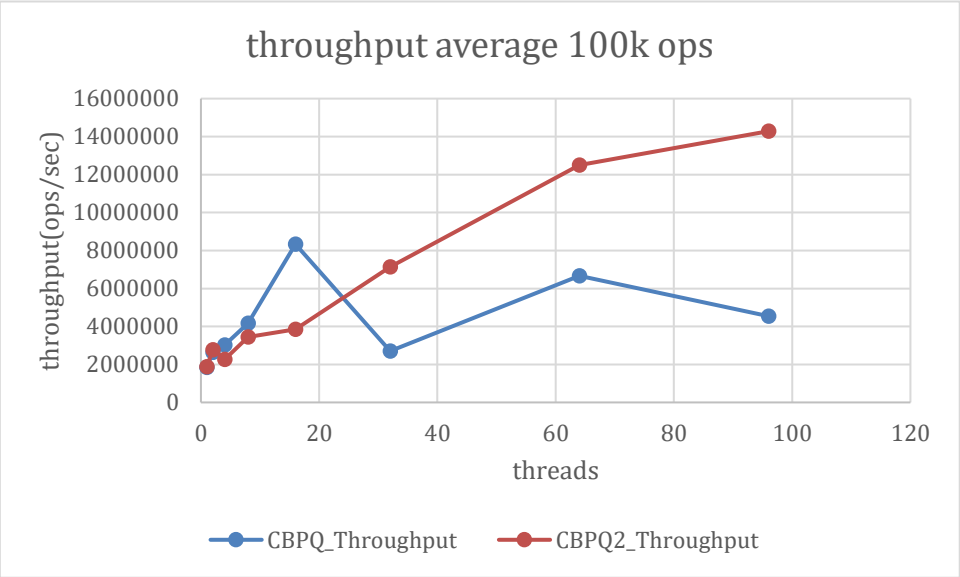




100 אלף הוצאות\הכנסות:

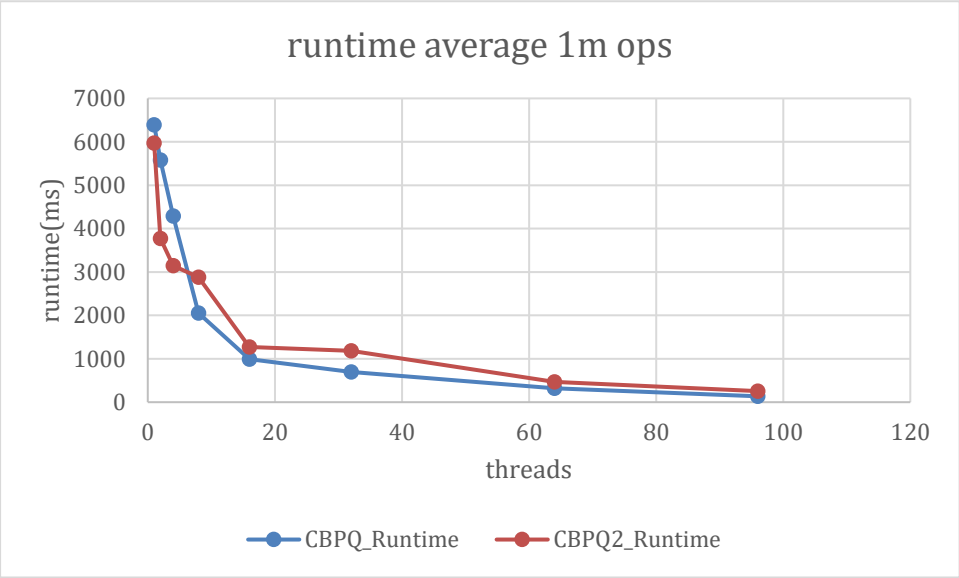
Threads	CBPQ_Runtime	CBPQ_Throughput	CBPQ2_Runtime	CBPQ2_Throughput
1	54	1851851.85	53	1886792.45
2	38	2631578.95	36	2777777.78
4	33	3030303.03	44	2272727.27
8	24	4166666.67	29	3448275.86
16	12	8333333.33	26	3846153.85
32	37	2702702.7	14	7142857.14
64	15	6666666.67	8	12500000
96	22	4545454.55	7	14285714.29

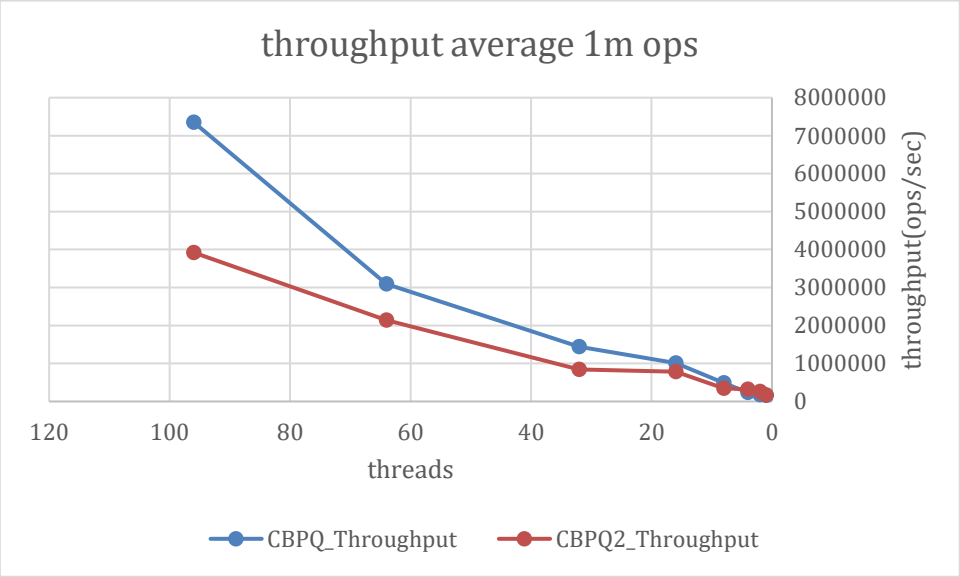




מיליון הוצאות\הכנסות:

Threads	CBPQ_Runtime	CBPQ_Throughput	CBPQ2_Runtime	CBPQ2_Throughput
1	6394	156396.62	5969	167532.25
2	5577	179307.87	3771	265181.65
4	4287	233263.35	3145	317965.02
8	2051	487567.04	2878	347463.52
16	991	1009081.74	1271	786782.06
32	695	1438848.92	1182	846023.69
64	323	3095975.23	467	2141327.62
96	136	7352941.18	255	3921568.63





מסקנות

מהתוצאות של הפרויקט והנתונים ניתן להסיק במספר נקודות עיקריות:

גרסת CBPQ2 המשופרת מציגה ביצועים טובים יותר בהשוואה ל CBPQ-המקורי בעיקר כשמספר החוטים עולה. הגרפים של הruntime מראים שCBPQ פחות טוב בזמן הריצה, בזמן ש CBPQ2נשאר יחסית עם זמני ריצה טובים יותר ויכול להצביע על מנגנון ניהול יעיל.

בגרף השני של התפוקה (throughput), CBPQ2 גם פה מציג שיפורים בביצועים לעומת CBPQ. אפשר להסיק שהמעבר לתור טבעתי בתוך כל chunks הפחית את הצוואר בקבוק ושיפר את הסקלאביליות.

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