|  |
| --- |
|  |
| מסדי נתונים מבוססי אחסון עמודות ושיטות לארגון עצמי של המידע השמור בהם |
| סמינר במערכות בסיסי נתונים |
|  |
| **מנחה: יעל ארד**  **סטודנט: עמיחי ימין**  **ת.ז: 303101547** |
| **תאריך הגשה: 18/2/2014** |

|  |
| --- |
|  |

תוכן עניינים

[1. מבוא 2](#_Toc380531050)

[2. מבנה ועיצוב ה-MonetDB 4](#_Toc380531051)

[2.1. מודל המידע הפיזי 4](#_Toc380531052)

[2.2. מודל עיבוד השאילתות 5](#_Toc380531053)

[2.3. ארכיטקטורת מערכת פנימית 7](#_Toc380531054)

[2.3.1. Front-end 7](#_Toc380531055)

[2.3.2. Back-end 7](#_Toc380531056)

[2.3.3. Kernel – גרעין המערכת 7](#_Toc380531057)

[3. נושאים מחקריים שהוטמעו ב-MonetDB 8](#_Toc380531058)

[3.1. מודעות למאפייני החומרה והרצה וקטורית 8](#_Toc380531059)

[3.2. שימוש חוזר בתוצאות ביניים 8](#_Toc380531060)

[3.3. עיבוד מתמשך של זרם נתונים 8](#_Toc380531061)

[4. מרחב הפיתרון לארגון-עצמי 9](#_Toc380531062)

[4.1. שינויים בצורת אחסון הנתונים 9](#_Toc380531063)

[4.2. שילוב בין רכיבי התוכנה של ה-MonetDB (ה-Software Stack) 10](#_Toc380531064)

[4.3. מודלי החלטה על קיטוע 12](#_Toc380531065)

[4.3.1. הקוביה המתפלגת נורמלית (The Gaussian Dice - GD) 12](#_Toc380531066)

[4.3.2. מודל העימוד המסתגל (Adaptive Pagination Model - APM) 13](#_Toc380531067)

[4.4. זמן ביצוע הקיטוע: שיטות לארגון עצמי 15](#_Toc380531068)

[4.4.1. קיטוע לאחר עיבוד השאילתא (Post-Processing) 15](#_Toc380531069)

[4.4.2. אחסון חמדן (Eager Materialization) 15](#_Toc380531070)

[4.4.3. אחסון עצל (Lazy Materialization) 15](#_Toc380531071)

[5. שיטת הקיטוע המסתגל 16](#_Toc380531072)

[5.1. אלגוריתם 16](#_Toc380531073)

[6. שיטת השכפול מסתגל 17](#_Toc380531074)

[6.1. אלגוריתם 18](#_Toc380531075)

[6.1.1. מציאת אוסף המקטעים הממשיים המינימלי לחישוב השאילתא - GetCover 18](#_Toc380531076)

[6.1.2. בדיקת מקטע לצורך שכפול - AnalyzeRepl 19](#_Toc380531077)

[6.1.3. בניית מקטעים ממשיים וחישוב תוצאת שאילתא - ScanMat 21](#_Toc380531078)

[6.1.4. הסרת מקטע ששוכפל במלואו – Check4Drop 21](#_Toc380531079)

[7. הערכה ניסויית 22](#_Toc380531080)

[7.1. סימולציה 22](#_Toc380531081)

[7.1.1. תקורת הארגון העצמי 22](#_Toc380531082)

[7.1.2. יתרונות הארגון העצמי 23](#_Toc380531083)

[7.1.3. גודל עץ השכפולים 25](#_Toc380531084)

[7.2. ניסוי בסביבה אמיתית 27](#_Toc380531085)

[8. סיכום 30](#_Toc380531086)

[9. ביבליוגרפיה 31](#_Toc380531087)

1. מבוא

בסיסי נתונים מבוססי אחסון עמודות נעשים נפוצים בשנים האחרונות, בעיקר בתחומי מחסני הנתונים, כריית מידע וביצוע אנליזות. ההבדל המרכזי בין בסיסי נתונים אלו לבסיסי נתונים קלאסיים (מבוססי אחסון שורות) הוא באופן האחסון – בעוד שבמסדי הנתונים הקלאסיים נתוני הטבלאות נשמרים ברשומות שלמות – אחת לאחר השניה, במסדי הנתונים מבוססי אחסון עמודות כל עמודה בטבלה נשמרת בנפרד. שיטה זו מאפשרת ביצוע יעיל יותר של פעולות אגרגטיביות על עמודות שלמות, כאשר מספר העמודות הנדרש כפלט קטן יחסית. אופי עבודה זה מתאים למערכות אפליקטיביות רבות בתחומים שפורטו קודם לכן.

בעבודה זו נציג דוגמה אחת למסד נתונים מבוסס אחסון עמודות - MonetDB, ובנוסף נציג שיטות לארגון-עצמי של המידע המאוחסן במסדי נתונים מסוג זה. שיטות אלו שולבו כהרחבה ב-MonetDB לצורך ביצוע הערכה ניסויית שלהן. העבודה כולה מבוססת על שני מאמרים שנכתבו במכון הלאומי למחקר בתחומי המתמטיקה ומדעי המחשב (CWI) הנמצא בהולנד, ושבו פותחה מערכת ה-MonetDB - [1], [2].

מערכת MonetDB היא מערכת קוד-פתוח חדישה לניהול מסדי נתונים מבוססי אחסון עמודות, המתאימה ליישומים עמוסי-ביצועים המבצעים תהליכי ניתוח על כמויות מידע גדולות. יישומים אלו עוסקים בתחומי כריית המידע, מודיעין עיסקי, עיבוד אנליטי מקוון (OLAP), מחקרים מדעיים, שאילתות XML, ואחזור מידע (טקסט ומולטימדיה). המערכת מפותחת מ-1993, ע"י קבוצת מחקר בנושא "ארכטיקטורות מסדי נתונים", במכון CWI.

המערכת עוצבה בראש ובראשונה עבור יישומי "מחסן נתונים" (Data Warehouse). יישומים אלו מאופיינים במסדי נתונים גדולים, כאשר רוב השאילתות שמופנות אליהם נדרשות לספק מודיעין עסקי או מידע תומך החלטות. שאילתות אלה הן בעלות אופי אנליטי – ביצוע חישובים סטטיסטיים על נתונים רבים כדי להסיק מגמות ושינויים, והן אינן שליפות של רשומות ספציפיות. לשם ביצוע השאילתות נדרש לסרוק חלקים נרחבים מבסיס הנתונים, ומכאן שיש חשיבות רבה לאחסון המידע באופן שיקל על חישובים וסריקות כאלו. מאפיין נוסף של יישומים אלו הוא שהמידע הקיים כמעט ולא מתעדכן, אך מידע חדש מתווסף באופן תדיר ובכמויות גדולות.

בימים אלו נעשה שימוש נרחב ב-MonetDB בתחומי הבריאות, התקשורת והמדעים. בנוסף, היא משמשת ללימוד וחקר ניהול המידע. המערכת מורדת מהאינטרנט מעל 10000 פעמים כל חודש.

MonetDB משיגה שיפורי-ביצועים משמעותיים בהשוואה לעיצובים "מסורתיים" של מסדי נתונים, באמצעות חידושים רבים בכל שכבות המערכת, וביניהם מודל אחסון המבוסס על שמירת עמודות בודדות, תהליך הרצת שאילתות חדיש המתואם ל-CPU, מפתוח מסתגל, ייעול שאילתות בזמן ריצה וארכיטקטורת תוכנה מודולרית.

בהמשך העבודה יוצג ה-MonetDB, נסביר על המבנה שלו ועל נושאים מחקריים מרכזיים שבאו לכדי מימוש.

השאיפה במערכות לניהול בסיסי נתונים, ובפרט ב-MonetDB, היא לייעל כמה שיותר את תהליכי ביצוע השאילתות. אחת מהדרכים לייעול הביצוע היא באמצעות "ארגון-עצמי" של המידע המאוחסן במסד הנתונים. הכוונה בארגון-עצמי היא לכך שמסד הנתונים מארגן-מחדש, מעדכן ומשנה בעצמו את צורת הארגון והאחסון של המידע השמור בו, בהתאם לעומס העבודה המשתנה. הרעיון של ארגון-עצמי אינו ייחודי למסדים מבוססי אחסון עמודות, אלא למסדי נתונים באופן כללי.

רעיון הארגון-העצמי הינו חיוני במיוחד למסדי נתונים הנדרשים לתמוך בעומס עבודה משתנה, כאשר כמות משאבי המערכת לניהול ותחזוקת הנתונים הינה מוגבלת. השאיפה היא לכך שתהליכי הארגון-העצמי של המידע יבוצעו תוך כדי ביצוע השאילתות הרגילות, כך שתיווצר תקורה מינימלית.

במסדי נתונים מבוססי אחסון שורות, ארגון מחדש של הנתונים משפיע בד"כ על האינדקסים והמבטים המאוחסנים (Materialized Views), שני רכיבים שאמורים להביא לשיפור בביצועי העבודה. עלות הבניה של שניהם היא גבוהה, והתרומה שלהם תלויה מאוד ביציבות של מאפייני העבודה השוטפת של מסד הנתונים. מכיוון שתחזוקת האינדקסים גם מהווה פעולה יקרה לביצוע, אלו שלא משפרים את ביצועי העבודה – מבוטלים. האתגר הוא לאזן בין הדרישות המתנגשות הללו – שיפור ביצועים אל מול תקורה ביצירה ובתחזוקה, ע"י בחירה מדוקדקת של העמודות מתוך הטבלה שימופתחו.

בניגוד לאחסון שורות, אחסון העמודות, שבו כל עמודה מאוחסנת עצמאית, פותח פתח להזדמנויות מפתוח חדשות ושונות, שכן ישנן פחות דרגות חופש במפתוח. כל עמודה היא מבנה נתונים בפני עצמו, שיכול להיות לו אינדקס או לא, ללא השפעה ישירה על מאפייני הגישה לעמודות אחרות. מצב זה לא קיים במערכות אחסון שורות, בהן אינדקס על עמודה מסוימת משפיע ישירות על מאפייני הגישה לעמודות האחרות בטבלה.

במצב של ארגון עצמי, החלטות מסוג זה צריכות להתקבל ללא התערבות אנושית. באופן אידיאלי, עומס השאילתות מנותח באופן שוטף, והארגון מחדש משולב יחד עם הרצת השאילתא. התרומה המרכזית של המאמר עליו מבוססת עבודה זו, היא חקר של הזדמנויות לארגון מחדש של הנתונים המאוחסנים, ארגון שמבוצע עצמאית ("ארגון עצמי") במסדי נתונים מבוססי אחסון עמודות. החקר והשלכותיו מוכוונים בעיקר לאופי עבודה מסוג "קריאה בלבד" של הנתונים, אופי עבודה שאופייני למערכות אנליטיות, ושעבורו אחסון עמודות מספק שיפורי ביצועים משמעותיים.

שיטות הארגון עצמי המוצגות בעבודה זו מתבססות על המושג "קיטוע". הכוונה בקיטוע היא חלוקה של עמודת נתונים מסוימת לחלקים קטנים יותר, כך שבמקום לסרוק את כל העמודה בעקבות תנאי בחירה מסוימים, ניתן יהיה להסתפק בסריקה של חלקים קטנים יותר המתאימים לתנאים אלו. תהליכי הקיטוע משתלבים בתוכניות ההרצה של השאילתות, על מנת לנצל את תוצאות הביניים של ההרצה ולהביא לתקורה מינימלית.

בעבודה נציג ונבחן שתי שיטות לארגון-עצמי במסדים מבוססי אחסון-עמודות:

* "קיטוע מסתגל" – שיטה המפצלת עמודה מסוימת למקטעים לא-חופפים לפי עומס וטווח הנתונים שבשימוש השאילות. פעולות הארגון מחדש והקיטוע מבוצעות על המידע באותו שטח האחסון. צורת עבודה זו מובילה לתקורה רבה בתחילת העבודה, שכן חלקים גדולים מהעמודה מטופלים ומחולקים במסגרת הארגון מחדש.
* "שכפול מסתגל" – שיטה המפצלת עמודה באמצעות שמירת עותקים של תוצאות שליפות, כדי לשפר שליפות עתידיות הפונות לתחום ערכים דומה. באופן זה הקיטוע מתקיים בשלבים: כאשר נקבע מקטע מסויים הוא מועתק למיקום אחר, והמקטעים שהוא גוזר על המקטע שפוצל יטופלו רק אם יהיה צורך (אם יגשו אליהם בהמשך). שיטה זו מובילה לדרישות אחסון גבוהות יותר מאשר קיטוע מסתגל, אבל מספקת תקורה נמוכה יותר.

המשך העבודה הינו כדלהלן: בפרקים 2 ו-3 נציג את מבנה ה-MonetDB ונושאים מחקריים הקשורים אליו. בפרק 4 נתעמק במרחב הפיתרון לארגון העצמי על רבדיו השונים. בפרקים 5 ו-6 נפרט על שתי שיטות הארגון העצמי, ובפרק 7 נציג הערכה ניסויית לשיטות הללו הן ע"י סימולציה והן באמצעות הרחבה של MonetDB. לבסוף, נסכם את העבודה ונפרט על היבטים למחקר עתידי בפרק 8.

1. מבנה ועיצוב ה-MonetDB

בפרק זה נציג את MonetDB, נתאר את הארכיטקטורה על חלקיה השונים ונפרט על מספר נושאים עיצוביים. היכרות עם מסד הנתונים חשובה להמשך העבודה, שכן ה-MonetDB מהווה מסגרת התייחסות לנושא הארגון העצמי ומשמש פלטפורמה לניסויים השונים שיוצגו בהמשך.

מנקודת המבט של המשתמש, MonetDB מהווה מסד נתונים רלציוני במלוא מובן המילה. המערכת תומכת בסטנדרט SQL:2003, ומספקת ממשקי לקוח סטנדרטיים כגון ODBC ו-JDBC, בנוסף לממשקי תכנות עבור שפות תכנות רבות כגון C, Python, Java, Ruby, Perl ו-PHP.

עם זאת, עיצוב ה-MonetDB מספק יכולות ייחודיות, שלא קיימות במסדי הנתונים הקלאסיים. בהמשך נפרט חלק מהן.

המערכת עוצבה כך שתוכל לנצל ביעילות את הזיכרון הראשי הגדול של המחשבים המודרניים במהלך עיבוד שאילתות, כאשר הנתונים עצמם שמורים באופן מתמיד על הדיסק הקשיח. העיצוב מתמקד בעומסי עבודה מדעיים ואנליטיים, שרובם קשורים לשליפות-מידע. מאפיין נוסף של עומסי העבודה הללו הוא שעדכוני מידע מהווים לרוב רק הוספה של מידע חדש למסד הנתונים, בכמויות גדולות בכל פעם, ולא עדכון של מידע קיים. עם זאת, MonetDB גם מספק תמיכה מלאה עבור טרנזאקציות, בהתאם לסטנדרט SQL:2003.

כלפי פנים, העיצוב, הארכיטקטורה ומימוש ה-MonetDB בוצעו תוך בחינה מחודשת של כל האספקטים והרכיבים של טכנולוגיות בסיסי-נתונים קלאסיות, על מנת שינצלו ביעילות את הפוטנציאל של החומרה המודרנית. בנוסף, MonetDB הוא אחד ממסדי הנתונים הראשונים המיועדים לציבור הרחב שמנצלים את טכנולוגיית אחסון העמודות. המערכת לא רק מאחסנת מידע לפי עמודות בודדות, אלא מביאה עמה גם עיצוב חדש לחלוטין של מנוע זמן הריצה, המותאם להרצת שאילתות עפ"י עמודות. מנוע זה עושה שימוש במבני נתונים ואלגוריתמים שמודעים לחשיבות זיכרון המטמון, ושמביאים לניצול אופטימלי של הזיכרון ההיררכי במערכות המודרניות.

בנוסף, MonetDB מספקת שיטות חדישות לתמיכה יעילה בעומסי-עבודה משתנים שאינם ידועים מראש, מעל נפחי מידע גדולים. השיטות הללו, וביניהן "מחזור" ו-"פיצוח מסד הנתונים", גורמות לתקורה מינימלית, ומביאות לתועלת גדולה בעיבוד עומס-העבודה והמידע ה"חם" בכל רגע נתון. "מחזור" היא שיטה גמישה לאחסון תוצאות שאילתות חלקיות בזיכרון המטמון, ו-"פיצוח מסד הנתונים" היא שיטת מפתוח שמשתנה יחד עם עומס העבודה.

העיצוב תומך גם בהרחבה של כלל המערכת, ברמות שונות. באמצעות מודולי הרחבה שמפותחים ב-C או בשפת MAL, ניתן להוסיף טיפוסי מידע ואלגוריתמים חדשים למערכת, כדי לתמוך בדרישות יישומיות ספיציפיות, שחורגות מתקן ה-SQL, או כדי לנצל ביעילות מאפייני מידע שקשורים לעולם תוכן ספציפי. כחלק מיכולת ההרחבה, MonetDB מספקת מעטפת מודולרית ורבת-שכבות לייעול-שאילתות, וגם היא ניתנת להרחבה ע"י חוקי-ייעול שקשורים לעולם תוכן ספציפי. יכולת ההרחבה הנ"ל משמשת להוספת יכולת ארגון-עצמי כפי שתוצג בהמשך.

בהמשך הפרק, נפרט על מודל המידע הפיזי של המערכת, מודל עיבוד השאילתות וארכיטקטורת המערכת הפנימית.

* 1. מודל המידע הפיזי

מודל האחסון ב-MonetDB שונה משמעותית מהמודל במסדים המסורתיים. במקום לאחסן את כל התכונות של תבנית יחס מסויימת ביחד כרשומה אחת (אחסון שורות), MonetDB מייצג תבניות יחס באמצעות פיצול אנכי (אחסון עמודות), כך שכל עמודה (או תכונה) שלמה של היחס מאוחסנת בנפרד בטבלה בעלת שתי עמודות – "מזהה אובייקט" (OID) ו-"ערך" (Value), הנקראית טבלת קישור בינארית או BAT (Binary Association Table). העמודה הראשונה בטבלה הזו, "מזהה אובייקט", מהווה את ראש הטבלה, בעוד העמודה השניה שמכילה את ערכי התכונה מהווה "זנב". ה-BAT הוא רכיב האחסון הבסיסי ב-MonetDB.

בצורה זו, כל טבלת יחס מיוצגת כלפי פנים בתור אוסף של BAT-ים. עבור יחס R עם k תכונות, יהיו קיימים k BAT-ים, כאשר כל BAT שומר תכונה בודדת של היחס בזוגות (OID, value).

מזהה האובייקט הוא שדה שמחולל אוטומטית ע"י המערכת, המזהה את רשומת היחס שהערך Value שייך אליה. מכאן משתמע שכל ערכי התכונות של רשומה מסוימת מותאמים לאותו מזהה האובייקט. מזהי האובייקטים יוצרים למעשה סדר בין ערכי עמודת-תכונה כלשהי, כך שמזהה מסויים מייצג את המיקום בעמודה של ערך של תכונה מסוימת. לכן, עבור BAT-ים בסיסיים, עמודת מזהה האובייקט אינה מאוחסנת בדיסק, אלא מתקבלת ישירות ממיקום הערך ב-BAT. דבר זה מביא לכך ש-BAT בסיסי זהה למעשה למערכים פשוטים מטיפוס כלשהו המוכרים לדוגמה משפת C. עבור כל רשומה t ביחס R, כל תכונות t מאוחסנות באותו המיקום בייצוגי עמודות ה-BAT המתאימות להן. המיקום נקבע עפ"י סדר ההכנסה של הרשומות. שיטת קביעת מיקום זו, והעובדה שהיא מתקיימת עבור כל העמודות, מאפשרת למערכת המבוססת אחסון עמודות לבנות מחדש רשומות ביעילות (בהינתן שהפעולות השונות על הרשומות שומרות על הסדר הנ"ל).

בהתאם לנאמר, עבור תכונות מטיפוסי מידע בגודל קבוע (כגון מספרים שלמים או ממשיים), MonetDB אכן משתמש במערכי C פשוטים מהטיפוס המדובר כדי לאחסן את עמודת ה-Value של BAT.

עבור תכונות מטיפוסי מידע בגודל משתנה (לדוג' מחרוזות), MonetDB מיישם סוג של קידוד מילוני. כל הערכים השונים האפשריים של עמודת-תכונה מסויימת מאוחסנים יחד ב-BLOB, ועמודת ה-Value ב-BAT היא למעשה מערך מטיפוס integer, המכיל הצבעות למיקומי הערכים האמיתיים ב-BLOB הנ"ל.

אחסון התכונות במערכים פשוטים כאלו מביא לכך שכלל איברי העמודה נמצאים פיזית באיזור רציף בזיכרון וללא חורים, דבר המאפשר לפצלם בקלות בכל נקודה שהיא.

מעבר לשיטת האחסון הפיזי, ישנם מספר נושאים נוספים המקבלים מענה במודל המידע הפיזי.

* מערכת ה-MonetDB משתמשת בקבצים ממופים בזיכרון (Memory Mapped File) של מ"ה, כדי לטעון מידע לזיכרון הראשי וכדי לממש זיכרון וירטואלי מורחב. מכאן הרי שכל מבני הנתונים מיוצגים באותו פורמט בינארי הן על הדיסק והן על הזיכרון הראשי.
* מערכת ה-MonetDB בונה מחדש רשומות מידע מאוחר ככל האפשר, כאשר במהלך כל עיבוד השאילתא, תוצאות הביניים הן בפורמט של עמודות (טבלאות BAT). רק לפני העברת התוצאה הסופית למשתמש, נבנות הרשומות הנדרשות. שיטה זו מאפשרת למנוע זמן הריצה לנצל, בזמן עיבוד השאילתות, פעולות ספציפיות המותאמות במיוחד למבני נתונים הדומים לוקטורים, שהן יעילות יותר מבחינת המעבד וזיכרון המטמון ביחס לפעולות על מבני נתונים אחרים.
  1. מודל עיבוד השאילתות

גרעין ה-MonetDB הוא מכונה מופשטת, שפותחה ב-MonetDB Assembly Language (MAL). ליבת ה-MAL מורכבת מאוסף פעולות אלגבריות יחסיות על BAT-ים הממומשות ברמה נמוכה (ראה דוג' באיור 1), ואשר באמצעותן השפה היא שלמה-חישובית. תוכניות הרצה המורכבות מפעולות אלגברה יחסית על רשומות שלמות (לדוג' ב-SQL) מתורגמות לפעולות אלגברה יחסית על BAT-ים, ומקומפלות לתוכניות MAL. תוכניות ה-MAL הללו מורצות לאחר מכן פקודה לאחר פקודה, כך שכל פקודה מחושבת על כל מידע הקלט עד לסיום, לפני שהפקודות העוקבות (שייתכן וקשורות לתוצאות הפקודה הקודמת) מורצות. שיטת ההרצה מבוססת על אחסון תוצאות חלקיות כפי שמתקבלות מפקודות מסוימות, כך שישמשו את הפקודות הבאות. בהמשך נראה שתוצאות חלקיות כאלו ישמשו את שיטות הארגון-העצמי כך שיתבצע במקביל להרצת השאילתא.

כל פעולה אלגברית על BAT-ים ממופה לפקודת MAL פשוטה, החסרה דרגות חופש מבחינת אופן פעולתה – היא לא מקבלת ביטויים מורכבים כפרמטר. פעולות מורכבות מפורקות לרצף פעולות אלגבריות על BAT-ים שכל אחת מבצעת תהליך יחיד פשוט על כל עמודת הערכים.

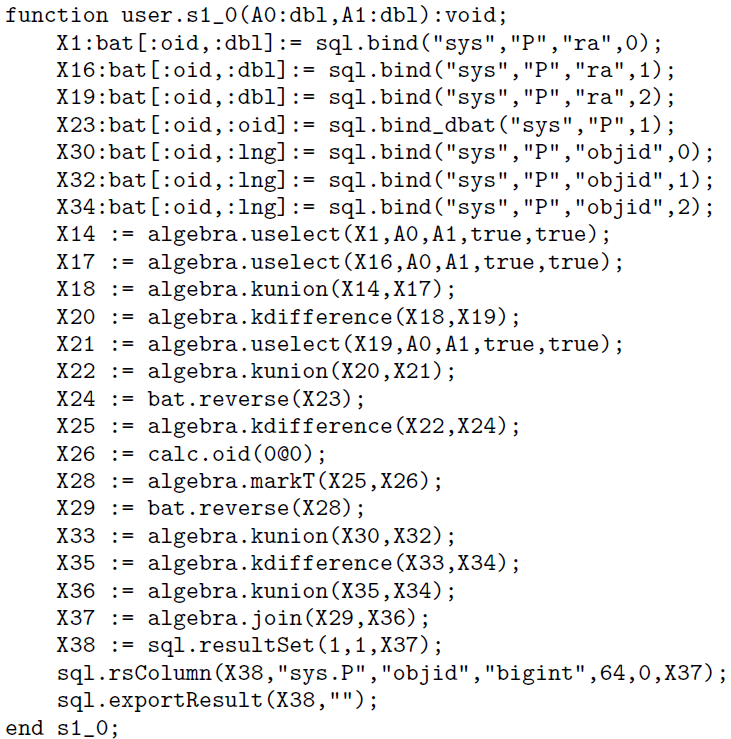
אופן עבודה זה מאפשר לממש את האלגברה על ה-BAT-ים תוך המנעות מהצורך לפרש את הביטויים עצמם – כל פעולה אלגברית על BAT ממומשת ע"י פעולה פשוטה על מערך. להלן מימוש לדוג' של פקודת select בשפת MAL, המתאימה לפעולה האלגברית של בחירת הערכים השווים ל-V:



לפעולות האלגבריות על ה-BAT-ים קיים יתרון חשוב – לולאות for הדוקות שלא מכילות קריאות לפונקציות, מספקות לוקליות גבוהה של פקודות, שכן כל פקודות הלולאה קרובות זו לזו במרחב הזיכרון. דבר זה פותר את בעיית ההחטאה בגישה לזיכרון המטמון של הפקודות (Instruction Cache Miss), ומביא להרצה מהירה יותר שלהן. לולאות פשוטות שכאלה מקובלות כאופטימיזצייה שמבצעים קומפיילרים.

באיור 1 מופיעה דוג' לתוכנית הרצה המתאימה לשאילתא הבאה ב-SQL:

SELECT OBJID FROM P WHERE RA BETWEEN 205.1 AND 205.12



איור 1: קוד MAL שמתאים לפקודת ה-SQL הבאה: SELECT OBJID FROM P WHERE RA BETWEEN 205.1 AND 205.12

* 1. ארכיטקטורת מערכת פנימית

סכמת עיבוד השאילתות ב-MonetDB מרוכזת בשלוש שכבות תוכנה:

* + 1. Front-end

השכבה העליונה, ה-Front-end, מספקת את מודלי המידע כפי שהמשתמש מכיר (טבלאות, XML, מערכים בהתאם לשיטת העבודה) ואת שפות השאילתות אליהן הוא רגיל (SQL, XQuery, SciQL, SPARQL – גם כן בהתאם לשיטת העבודה). משימת שכבה זו היא למפות את מודלי המידע כפי שמוכרים למשתמש לאוסף BAT-ים, ולתרגם את השאילתות בשפות השונות לשאילתות בשפת MAL.

בתחילה, שאילתה (לדוג' ב-SQL) מומרת לייצוג פנימי של אלגברה יחסית (תוכנית ריצה לוגית), ומבוצעות עליה אופטימיזציות בהתאם לעולם-תוכן ספציפי. אופטימיזציות אלו נועדו בראש ובראשונה להפחית את כמויות המידע שיש לעבד ובהתאם גם להקטין את תוצאות הביניים.

לאחר מכן, תוכנית הריצה הלוגית המיועלת מקומפלת לשפת MAL, ומועברת לשכבה הבאה לטובת ייעול MAL כללי וביצוע.

* + 1. Back-end

השכבה האמצעית, ה-Back-end, מורכבת ממעטפת מייעלי MAL (נקרא למעטפת זו בשם הכולל – "המייעל הטקטי") וממפרש שפת MAL עבור השכבה התחתונה.

המייעל הטקטי מורכב מאוסף מודולי ייעול, שכל אחד מהם ממיר תוכנית MAL נתונה לאחת יעילה יותר. ייעול טקטי זה שואב השראה בעיקר מתהליכי ייעול של שפות תכנות ופחות מתהליכי ייעול שאילתות במסדי נתונים קלאסיים.

השימוש במייעל נעשה כדי לטייב את תוכנית ההרצה בעזרת מידע על זמינות משאבים וחוקים נוספים. בנוסף, לכל פקודה יחסית המופיעה בתוכנית ההרצה קיים מודול ייעול ייעודי שבוחר את האלגוריתם המתאים ביותר עבור האופרנדים בפעולה. שיטה זו של מודול ייעול ספציפי לפעולה היא אחת הטכניקות המרכזיות המביאות ליעילות של מערכת מבוססת MonetDB.

באיור 1 ניתן לראות תוכנית הרצת שאילתא לא אופטימלית. לאחר ביצוע האופטימיזציה במייעל הטקטי, התוכנית המלאה מכילה כ-80 פקודות MAL, כולל פקודות לניהול משאבי מערכת, כמו שחרור BAT-ים של תוצאות ביניים.

מכיוון ששכבה זו פועלת על האלגברה היחסית המשותפת שמגיעה מהשכבה העליונה, מודולי האופטימיזציה הללו משותפים לכל שפות השאילתות ומודולי המידע שבשכבה העליונה.

* + 1. Kernel – גרעין המערכת

השכבה התחתונה, ה-Kernel, מספקת את מבני ה-BAT-ים, בתור מבני הנתונים איתם ה-MonetDB עובדת, ובנוסף מספקת ספרייה של מימושים יעילים לפעולות בינאריות של אלגברה יחסית (לדוג' פעולת ה-select שהופיעה בסעיף הקודם). שכבה זו מהווה למעשה את מנוע זמן הריצה של המערכת, והיא זו שמריצה את תוכניות ה-MAL הסופיות כולל הבאת הנתונים מהדיסק לזיכרון.

עקב שיטת העיבוד הפשוטה של פקודה לאחר פקודה, בלי ניתוח אמיתי של הקלט, לכל פקודה ב-MAL יש גישה לכל הקלט שלה כולל המאפיינים שלו. דבר זה מאפשר לפעולות האלגבריות לבצע אופטימיזציות זמן-ריצה, ע"י בחירת האלגוריתם והמימוש המתאימים ביותר לשימוש בהתאם למאפייני הקלט.

לדוג', פעולת select יכולה לנצל BAT ממויין מראש ע"י הפעלת חיפוש בינארי או לנצל אינדקס קיים עבור שליפות נקודתיות או לבצע סריקה מלאה בכל מקרה אחר. באופן דומה, פעולת join יכולה להחליט בזמן-ריצה אם לבצע merge-join במידה והקלטים ממוינים, או לעבור ל-hash-join בכל מקרה אחר.

1. נושאים מחקריים שהוטמעו ב-MonetDB

בסעיף זה נפרט בקצרה על מספר נושאי חקר מעולם אחסון העמודות, בהקשר של MonetDB.

נושאים אלו, יחד עם רבים נוספים, ממומשים כחלק מגרעין ה-MonetDB.

* 1. מודעות למאפייני החומרה והרצה וקטורית

חידוש מרכזי ב-MonetDB הוא ההסתמכות על אלגורימים מודעי-חומרה. בעשורים האחרונים, השיפורים במהירות המעבדים עקפו את השיפורים בזמני ההשהיה בזכרונות ה-RAM. בהתאם לכך, הגישה לזיכרון הראשי הפכה לצוואר בקבוק עבור יישומים רבים, כולל מערכות לניהול מסדי נתונים.

על מנת להתגבר על בעיה זו, יש להשקיע מאמצים בניצול טוב של זיכרון המטמון של המעבד (CPU cache), ולשם כך נדרש להביא לייעול מקס' בתבניות הגישה לזיכרון. באופן זה, התפתחו מספר אלגוריתמי עיבוד שאילתות, במיוחד בתחומי ה-join, שבאים לענות על דרישה זו, ומשולבים ב-MonetDB.

הרעיון המרכזי בהם הוא הגבלת הגישות האקראיות לזיכרון לאזורי מידע שמתאימים בדיוק לזיכרון המטמון של המעבד, וכך לחסוך בהחטאות זיכרון מטמון (cache miss) ולשפר את הביצועים.

בנוסף, ההגבלה לאיזורי מידע כאלה, יחד עם שיטת ההרצה הוקטורית (פקודה אחת בכל פעם, הפועלת על מערך, כפי שפורט קודם לכן) מביאה לכך שתוצאות ביניים לא יצטרכו להישמר בזיכרון הראשי, אלא מתאימות גם הן לאחסון בזיכרון המטמון. דבר זה גם מביא לשיפורי ביצועים משמעותיים.

* 1. שימוש חוזר בתוצאות ביניים

שיטת העיבוד במסדי נתונים מבוססי אחסון עמודות מחייבת יצירה ואחסון של תוצאות ביניים. פרוייקט ה-"ממחזר" (Recycler) ששולב ב-MonetDB, מאחסן ומשתמש מחדש בתוצאות הביניים הללו. לדוג', במקרה שפעולת select מכוסה כבר ע"י תוצאת ביניים מאוחסנת של שאילתא ישנה, המערכת תעדיף להשתמש בה במקום לעבד מחדש את העמודה המקורית.

תוצאות ביניים נשמרות כל עוד קיים מקום פנוי בזיכרון המוקצה ל-"ממחזר", כאשר העדיפות היא למידע "חם" – מידע שנעשה בו שימוש רב יחסית.

* 1. עיבוד מתמשך של זרם נתונים

יישומים רבים המפותחים כיום הינם יישומים מקוונים, בעלי קלט שהוא למעשה זרם נתונים מתמשך (Streaming). יישומים אלו מקבלים כל הזמן מידע חדש, וצריכים לעבד ולנתח אותו בזמן קצר מאוד לפני שהמידע הבא יגיע. תרחישים כאלו מתקיימים במערכות ניטור רשתות, במערכות רישום log, במערכות פיננסיות ובעוד רבות.

כחלק מהמחקר סביב ה-MonetDB, פותח מנוע עיבוד לזרם נתונים מתמשך (Stream engine), כהרחבה ל-MonetDB, המתמקד בניתוח מעמיק תוך כדי שהמידע מגיע. האתגרים המרכזיים בפיתוח מנוע שכזה הם עיבוד יעיל בחלון זמן מוגבל, זמני תגובה מהירים למידע הטרי ויצירת אינדקסים דינמיים עבור מידע שהולך ומשתנה.

1. מרחב הפיתרון לארגון-עצמי

פיתרון לארגון-עצמי חייב לתת מענה למספר נושאים עיצוביים. פרק זה עוסק בחשובים שבנושאים הללו: שינויים נדרשים בצורת אחסון הנתונים, בחירת הרכיב המתאים ביותר לשילוב הפיתרון מתוך רכיבי מסד הנתונים, האופנים על-פיהם נקבע אם צריך לבצע ארגון-מחדש של הנתונים והשלב שבו מבצעים את הארגון-מחדש עצמו.

יש להדגיש ששיטות הארגון העצמי מתאימות למסדי נתונים מבוססי אחסון עמודות באופן כללי, ולא רק ל-MonetDB. עם זאת, מבחינת המימוש, נשים דגש על השילוב ב-MonetDB, מכיוון שהוא מהווה פלטפורמה ניסויית לשיטות.

* 1. שינויים בצורת אחסון הנתונים

מסדי נתונים קיימים בתחום אחסון העמודות מגדירים בד"כ קריטריון קשיח לקיטוע. לדוגמה, יכול להיות קריטריון שדורש מכל עמודה להיות מיוצגת ע"י רצף בלוקים בגודל 64KB בלבד. הגישה לנתונים בפעולות השונות מתבצעת על בלוק אחד בכל פעם. בנוסף, מכיוון שבאחסון עמודות נדרש לייצר רשומות נתונים מתוך עמודות עצמאיות, השיטה הנפוצה לארגון הפיזי של העמודות בבלוקים היא סידור לפי מספר הרשומה אליה שייך הנתון, קרי רכיבי העמודות של שורה i יופיעו במיקום אחיד בכל אחסוני העמודות, דבר המאיץ את בניית הרשומה מאוסף העמודות. כפי שראינו קודם, מערכת ה-MonetDB משתמשת גם היא בצורת ארגון פיזי זאת. יחד עם זאת, לשיטה זו ישנה גם משמעות בעייתית – פעולות שנמצאות בסוף תוכנית ההרצה של השאילתא (Query Execution Plan), בד"כ פעולות בחירה (Where Clauses), דורשות גישה לכל העמודה המאוחסנת בדיסק, שכן הסידור בעמודה אינו לפי הערך הנבחר אלא לפי מספר הרשומה אליה הוא שייך.

אל מול צורת ארגון וקיטוע זו, ועל מנת לספק תשתית לשיטות הארגון העצמי, מוצעת צורת ארגון עמודות חלופית – ארגון עמודות מבוסס-ערכים: עמודה מחולקת לאוסף מקטעים, כך שכל מקטע מכסה טווח רציף של ערכים אפשריים. ארגון כזה מאפשר לתחזק אינדקס דליל של המקטעים, כך שמייעלי השליפות בוחרים בעזרתו רק את מקטעי העמודה שחופפים עם פסוקי הבחירה בשאילתא (Where Clauses). בדומה לעבודה עם בלוקים, הרצת השאילתא מתבצעת ע"י מעבר על המקטעים השונים – מקטע אחד בכל פעם. ההבדל הוא שכעת אין צורך בגודל אחיד וקבוע מראש של כל מקטע. במקום זאת, הקיטוע האידיאלי הוא זה שמייצג את פיצול טווח ערכי העמודה ע"י עומס העבודה (השאילתות המבוצעות), לתוך אוסף תתי-טווחים. לרוע המזל, עומס העבודה לא ידוע מראש ואף ייתכן שהוא משתנה עם הזמן, ולכן קשה לחזות ולהשיג את הקיטוע האידיאלי. עם זאת, המערכת יכולה להשתמש במאפייני עומס העבודה ובנתוני השאילתות המורצות, כדי לייצר מקטעים עבור חלקי-עמודות שהשאילתות השונות פונות אליהם.

לארגון מבוסס-ערכים ישנם יתרונות גם בהיבטי שטח האחסון הנדרש. אינדקס דליל של מקטעים דורש שטח אחסון מוגבל וקטן, ועדיין מייעל את הגישה למקטעים. בהשוואה לכך, כדי להאיץ את השליפות בעזרת מפתוח של ארגון מבוסס-מיקום נדרש אינדקס גדול ודחוס, שבמקרים מסויימים אף עלול להגיע לעותק שלם של העמודה עצמה. ארגון מבוסס ערכים יכול גם להשתלב עם אלגוריתמי דחיסה על עמודות.

למרות היתרונות השונים, לארגון מבוסס-ערכים קיימים גם חסרונות. מכיוון שההתאמה במיקום של הערכים בעמודות השונות לא נשמר, פעולות המבוססות על ההתאמה הזו, לדוג' בניית רשומה מהערכים בעמודות שונות, עלולות להתבצע בזמן ארוך יותר. ההתנגשות בין היתרונות של שיטת הארגון הזו (בהיבט של שליפה לפי טווח ערכים) אל מול התקורה שיכולה להגרם לפעולות אחרות (לדוג' בניית רשומה) משמשת כקריטריון בזמן עיצוב פיסי של מסד נתונים מבוסס אחסון-עמודות.

* 1. שילוב בין רכיבי התוכנה של ה-MonetDB (ה-Software Stack)

הנושא העיצובי המרכזי של מערכת מאורגנת-עצמאית הוא באיזה רכיב תוכנה יש לשלב את ההחלטות על קיטוע מידע. החלטות מסוג זה נפוצות כבר היום – חלקן קיימות כחלק מקוד המערכת כגון גודל קבוע של דף מידע, וחלקן קיימות כחלק מסכמת הנתונים הלוגית – לדוג' חוקים לחלוקת המידע המהווים חלק מהגדרת הטבלה, כפי שהמשתמש מספק.

מטרת המחקר שבוצע, ייצור חוקים לניהול עצמי של המידע, נופלת בקטגוריה השניה. עם זאת, ארגון-מחדש לא אמור להשפיע כלל על משתמש הקצה, שלא אמור להיות מודע להחלטות הללו ולא מסוגל לשלוט עליהן ישירות. מסיבה זו, לא ניתן לשלב החלטות לארגון מחדש כחלק מהמידע שהמשתמש מספק וכחלק משכבת ה-Front-end.

אם כך, אפשרות ראשונה לשילוב היא ב-Kernel, במנוע זמן הריצה. ב-MonetDB כל אחת מהפעולות מיועלות ליצירת תוצאות ביניים תוך שימוש במסגרת זיכרון ראשי המוגדרת מראש. לכן, מימוש קיטוע מידע ברמה הזו של המערכת ידרוש עיצוב מחדש של המנוע, דבר שישפיע על אלפי אלגוריתמים.

בנוסף, בעולם אידיאלי, הקיטוע הטוב ביותר הוא זה שצופה ותומך באופי העבודה השוטפת, בצורה הטובה ביותר. מכיוון שהעתיד קשה לחיזוי, ניתן להשתמש בעבר הקרוב בתור חיזוי מספק. פיתרון שמוגדר על בסיס הגדלים הקבועים מראש של בלוקי הזיכרון (לדוג', האיברים עליהם המנוע יודע לעבוד), לא ייחשב פיתרון מתאים לסביבות תוכנה בעלות עומס עבודה משתנה.

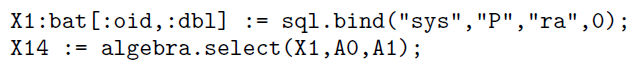
מהסיבות הללו, ארגון-עצמי צריך להיות באיזור המייעל הטקטי של ה-MonetDB, הרכיב בו מתקבלות החלטות לגבי משאבים גלובליים ותוכניות MAL כלליות מומרות לכאלה שמטפלות במקרים ספציפיים. הטיפול בארגון העצמי יתבצע ע"י זיהוי BAT-ים שקוטעו, ושליחת קריאות ל"מייעל קיטוע", שימיר פעולות MAL שמבוצעות על BAT מקוטע – לאוסף פעולות שמודעות לקיטוע, שיופעלו רק על המקטעים שרלוונטיים לשאילתא.

קיימות שתי שיטות להחלפת הקוד, כשההחלטה על השיטה המתאימה תלויה במספר המקטעים של BAT מסויים, נתון שעדיף שיהיה ידוע בזמן ייעול השאילתא. עבור מספר קטן של מקטעים, מופע של הפעולה מתווסף עבור כל מקטע שרלוונטי לשאילתא. עבור מספר גדול של מקטעים, מתאימה יותר גישת האיטרציה על המקטעים. בשתי השיטות, אוסף פקודות נוסף נדרש לטובת האיחוד של תת-תוצאות לתוצאה אחת גדולה.

מייעל הקיטוע גם אמור לאסוף מידע על ניצול הזיכרון והמעבד על-ידי הפעולות המבוצעות על BAT מסויים, ועל-פיו להחליט אם לקטע אותו או לאחד מספר מקטעים יחד. ההחלטה לשלב את תהליך הקיטוע במייעל הטקטי היא קריטית לביצוע ארגונים מחדש תוך כדי העבודה הרגילה של המערכת.

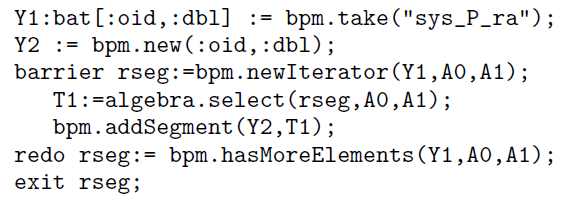
מייעל הקיטוע משתמש באינדקס מקטעים שנמצא בזיכרון הראשי, כדי לגלות בקלות טבלאות מקוטעות בתוכנית השאילתא. אינדקס זה גם מספק מידע נלווה על המקטעים, שהמייעל יכול לעשות בו שימוש. לדוג' באמצעות מידע על גודלי המקטעים, המייעל יכול לשפר את הקצאות הזיכרון הנדרשות לביצוע תוכנית השאילתא.

כדי להציג קוד שנוצר באמצעות מייעל הקיטוע, נשתמש בפעולת הבחירה הראשונה מהדוג' באיור 1. קטע הקוד הרלוונטי מתוך תוכנית ההרצה הוא:



המשתנה X1 נקשר ל-BAT בשם RA, והפרמטרים A0,A1 נקשרים לקבועים 205.1, 205.12 בהתאמה (עפ"י שאילתת ה-SQL המתאימה). פעולת הבחירה (SELECT) מוצאת את הנתונים שמתאימים לטווח שהופיע בפקודת ה-SQL : RA BETWEEN 205.1 AND 205.11. עבור RA BAT שאינו מקוטע ומסודר לפי מיקום, ייבחר מימוש של פעולת הבחירה שמבוסס על סריקה מלאה.

ברגע ש- RA BAT מפוצל למקטעים מבוססי-טווח, מייעל הקיטוע ימיר את קטע הקוד הנ"ל לרצף פקודות המשלבות איטרציה על המקטעים השונים:



המשתנה Y1 מקושר ל-RA BAT המפוצל למקטעים מבוססי-טווח, והמשתנה Y2 מקושר ל-BAT מקוטע חדש שיחזיק את תוצאות הבחירה. האיטרטור מודע לטווח אותו יש למצוא, ומשתמש באינדקס המקטעים כדי לעבור רק על המקטעים שחופפים עם טווח החיפוש [A0,A1].

בנוסף להחלפת הפקודות לכאלה שמודעות לקיטוע, קיימים מודולים שלמים הקשורים לקבלת ההחלטה על הקיטוע ולביצוע הפעולה עצמה – הארגון מחדש של העמודה. מודולי החלטה וארגון מחדש אלו ממומשים בהתאם לתיאור בשני הסעיפים הבאים.

בזמן ביצוע השאילתא, מנוע זמן הריצה יריץ את קוד ה-MAL החדש וכן את מודולי הקיטוע, וכך יתבצע הארגון-מחדש תוך כדי הרצת השאילתות.

* 1. מודלי החלטה על קיטוע

בעוד שהחלטות קבועות על קיטוע מבוססות על הנחות לגבי עומס ואופי עבודה יציבים וצפויים, קיטוע בארגון-עצמי מתאים את עצמו לעומס ולאופי העבודה באופן שוטף. בהינתן פסוק בחירה על עמודה מסויימת, המערכת צריכה להתנהג עפ"י מודל כלשהו, כאשר היא צריכה להחליט אם להשתמש בתוצאת הבחירה כדי לייצר מקטע חדש או לא.

אינטואיטיבית, המודלים השונים צריכים להמנע מיצירת כמות גדולה של מקטעים קטנים, כדי לשמור על תקורה מינימלית בתחזוקת מקטע וכדי למנוע איטרציה ארוכה מדי על מקטעים, כאשר מוצאים אילו מקטעים חופפים לבחירה מסוימת. מצד שני, מקטעים גדולים מדי מביאים לקריאות מיותרות מהדיסק ולצריכת זיכרון גבוהה בזמן ביצוע השאילתא.

לכן, יש צורך למצוא מודלים שמכוונים לא רק לקיטוע העמודה במקומות יעילים מבחינת השאילתות שמורצות, אלא גם למציאת איזון מתאים בין גודלי המקטעים.

נציג שני מודלי החלטה על קיטוע שעונים על שתי הדרישות הללו: "הקוביה המתפלגת נורמלית" ו-"מודל הדפדוף המסתגל".

* + 1. הקוביה המתפלגת נורמלית (The Gaussian Dice - GD)

המודל משתמש בהתנהגות אקראית כדי לקבל החלטות על קיטוע. תבנית נאיבית למודל תהיה הטלת מטבע עבור כל שאילתא, ועל-פיו להחליט אם להמשיך ולחלק את העמודה הנוכחית למקטעים נוספים. כדי להמנע מיצירת מקטעים קטנים מאוד, נשתמש ב-"אקראיות לומדת", שמשקפת את המצב המשתנה של המקטע לאורך הזמן. המטרה באקראיות שכזו היא לתת עדיפות לפעולות שמחלקות את המקטע לחלקים שווים בקירוב, ולהפחית את ההשפעה של שאילתות נקודתיות על מבנה המקטע.

המודל משתמש בשני יחסים מרכזיים:

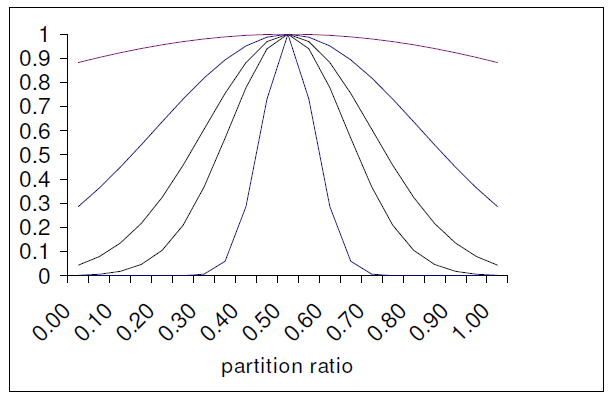
* A: היחס בין גודל המקטע שעתיד להיווצר (P) וגודל המקטע שצריך להחליט לגביו (S).
* B: היחס בין גודל המקטע שצריך להחליט לגביו (S) וגודל העמודה השלמה (Total).

הבסיס למודל הוא ההתפלגות הנורמלית G, כאשר התוחלת היא μ=0.5 וסטיית התקן היא σ=SizeS/SizeTotal.

הקביעה לגבי μ מובילה לכך שההסתברות הגבוהה ביותר להחלטה חיובית היא כאשר תוצאת הקיטוע תוביל לשני תת-מקטעים שווים בגודלם (SizeP/SizeS=0.5).

הפונקציה O(x) = G(x)/G(0.5) משמשת כפונקציית החלטה לגבי כל היחסים האפשריים. החלוקה ב-G(0.5) היא למעשה נרמול, המביא לכך שעבור יחס SizeP/SizeS של 0.5, ההסתברות תמיד תהיה 1.

צורת ההתפלגות של O(x) עבור מספר ערכי σ מופיעה באיור 2.



**יחס החלוקה A**

איור 2: התפלגות ההסתברות לקבלת החלטה על קיטוע במודל "הקוביה המתפלגת נורמלית", עבור מספר ערכי σ.

המודל פועל באופן הבא: בכל פעם שיש לקבל החלטה על חלוקה של מקטע מסויים, בוחרים אקראית מספר r∈[0..1) ובודקים אם מתקיים r<O(SizeP/SizeS). באופן זה, עבור שליפות שמפצלות מקטע ביחס של x=0.5, קיימת סבירות גבוהה יותר שישתמשו בהן לצורך ארגון-מחדש, לעומת פעולות שיוצרות מקטעים קטנים יחסית.

נסביר את הקביעה לגבי הפרמטר σ. קביעה זו נותנת עדיפות לשאילתות שמביאות לחלוקה של מקטעים גדולים יחסית. למעשה ככל שהמקטע שלגביו צריך להחליט על קיטוע מהווה חלק גדול יותר מהעמודה השלמה, ערך ה- σיהיה גבוה יותר, דבר שיוביל לגרף מתון יותר, ולכן להסתברויות גבוהות יותר גם עבור יחסי SizeP/SizeS שאינם קרובים ל-0.5. כלומר, בהנתן שהיחס B גדול יותר – קיים סיכוי רב יותר שהמקטע המדובר יפוצל, ללא תלות ביחס A.

* + 1. מודל העימוד המסתגל (Adaptive Pagination Model - APM)

במודל זה ההחלטה על ארגון-מחדש מתקבלת באופן דטרמיניסטי באמצעות קירובים על גודלי מקטעים. במטרה לייצר איזון בין גודלי המקטעים השונים, נציג שני גבולות המתייחסים לגדלים הללו, הנמצאים בשימוש במודל:

* הגבול התחתון (Mmin) שומר על המערכת מפני חלוקה למקטעים קטנים מדי.
* הגבול העליון (Mmax), שבהכרח גדול מהגבול התחתון, מייצג את גודל המקטע המקסימלי לגביו המערכת מסכימה לבצע פעולות יקרות יותר על שאילתות נקודתיות במקום לפצלו למקטעים קטנים מדי.

להלן החוקים על-פיהם המודל מחליט לגבי פיצול מקטע מסויים S:

1. אם מתקיים SizeS<Mmin, המקטע נותר ללא שינוי.
2. אם מתקיים SizeS>Mmin, נבדקים הגדלים של התת-מקטעים שייוצרו באמצעות השליפה הנוכחית. אם הגודל המקורב של כולם גדול מ-Mmin, המקטע המקורי S יאורגן-מחדש באמצעות התוצאה המאוחסנת של השליפה.
3. אם השליפה יוצרת תת-מקטעים שקטנים מ-Mmin, אזי המקטע לא מתאים לפיצול כרגע.

עם זאת, ישנו סיכוי שתתבצע שליפה נוספת שמערבת את המקטע S בעתיד הקרוב. לכן, כדי לזרז שליפות עוקבות וכדי לשלוט בצורה טובה יותר על צריכת הזיכרון – המקטע יאורגן מחדש במידה ו- SizeS>Mmax.

מכיוון שגבולות השליפה במצב זה יגרמו ליצירת מקטע קטן מדי, נקודת הפיצול תיבחר מבין שני גבולות השליפה או קירוב של הערך הממוצע במקטע.

מודל ה-APM מאופיין בכך שגודלי המקטעים שהשאילתות השונות פונות אליהם מתכנסים יחסית מהר לטווח הבא: Mmin<=SizeS<=Mmax. ע"י שינוי של הפרמטרים Mmin ו-Mmax, ניתן לכוונן את המודל כך שיתקבלו יותר או פחות החלטות על ביצוע ארגון מחדש של עמודות.

* 1. זמן ביצוע הקיטוע: שיטות לארגון עצמי

הנושא העיצובי השלישי הוא זמן ביצוע הקיטוע עצמו. כאשר מייעל הקיטוע מחליט שמשתלם לפצל מקטע S, כל שנותר הוא להחליט מתי לבצע את הקיטוע.

קיימות שלוש חלופות. מעבר להחלטה על זמן הביצוע, כל חלופה מהווה שיטה לארגון-עצמי בפני עצמה. נציג את שלושתן בקצרה ולאחר מכן נתמקד בשתיים המרכזיות.

* + 1. קיטוע לאחר עיבוד השאילתא (Post-Processing)

מייעל הקיטוע רק מסמן את המקטע המיועד לקיטוע, והארגון-מחדש מבוצע לאחר סיום ביצוע השאילתא. שיטה זו מאפשרת מציאת "נקודה אידאלית" עבור הקיטוע, לדוג' נקודה שתביא ליצירת תת-מקטעים שווי גודל כדי לאזן את השימוש במשאבי הזיכרון. מעבר לכך, בשיטה הזו אפשר לאחד מספר פיצולים מוצעים לפעולה ארוכה אחת, כך שייבחרו נקודות פיצול מיטביות.

עם זאת, העיכוב שעלול להגרם לאחר ביצוע השאילתא, עלול לגרום לכך שהשאילתות הבאות שיתעסקו באותו המקטע יפספסו שיפורים אפשריים. בנוסף, התקורה המלאה של הארגון העצמי יכולה להיות יקרה יותר, מכיוון שכדי לקבל החלטות על הקיטוע יש צורך להביא לזיכרון את המקטעים המסומנים ולסרוק אותם מחדש.

חלופה זו דומה לשיטות ארגון מחדש לא-עצמאיות במערכות רבות. בעבודה זו לא נעמיק בה.

* + 1. אחסון חמדן (Eager Materialization)

מייעל הקיטוע שומר תת-מקטע שנוצר בזמן השליפה, ומארגן מחדש את המקטע המקורי בשלב מוקדם, ולא לאחר סיום ביצוע השאילתא.

לכן, יצירה ואחסון של תת-מקטע המתאים לשליפה מבוצעים תוך כדי הרצת השאילתא, ולא מביאים לעלויות נוספות. עם זאת, התת-מקטע הנ"ל מפצל את המקטע המקורי ומביא ליצירת תת-מקטעים נוספים (אלו שמעבר לטווח השליפה) שיש לאחסן גם אותם. .

פעולה זו גורמת לתקורה המרכזית של השיטה. התקורה הזו מביאה להארכת זמן העיבוד של שאילתות מוקדמות יותר בחיי המערכת, כאלו שמביאות לארגון-מחדש של עמודות שלמות או של חלקים גדולים מהן. זמן העיבוד מתקצר ככל שיותר ויותר שאילתות מורצות, שכן התקורה של קיטוע מקטעים גדולים הולכת וקטנה. נקודה בעייתית נוספת בשיטה זו, היא שגבולות השליפה אינן תמיד נקודות הפיצול האופטימליות מבחינת משאבי הזיכרון.

נקרא לשיטה הזו בשם "קיטוע מסתגל", ונרחיב לגביה בפרק 5.

* + 1. אחסון עצל (Lazy Materialization)

התוצאות שמתקבלות משליפה מסוימת (שהן למעשה תת-מקטעים חדשים) נשמרות כמו בשיטה הקודמת, ונרשמות באינדקס המקטעים, אבל החלקים הנותרים של המקטע המקורי יאורגנו-מחדש רק בעת הצורך, כאשר שאילתות עתידיות יפנו אליהם.

בצורה זו, נשמרים תת-מקטעים משוכפלים וחלקיים של המקטע המקורי, ללא מחיקת המקטע המקורי, דבר שמוביל לדרישות אחסון גבוהות יותר. בנוסף, שיטה זו משפיעה על זמן הביצוע של שאילתא, כי צריך למצוא את סט העותקים המתאים ביותר לדרישות השאילתא.

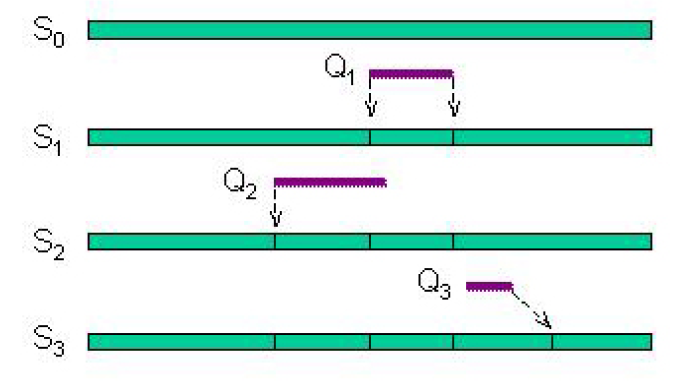
היתרון המרכזי הוא שכמעט כל הארגון-מחדש מבוצע תוך כדי הרצת השאילתא, דבר המביא לתקורה מינימלית ולהפרעה מינימלית לעומס העבודה.

נקרא לשיטה הזו בשם "שכפול מסתגל", ונרחיב לגביה בפרק 6.

בעבודה זו נציג את שתי השיטות האחרונות לארגון-עצמי. שיטות אלו ממומשות כמודולי ארגון-עצמי ומשתלבות, בעזרת מייעל הקיטוע, בתהליכי עיבוד והרצת השאילתא.

1. שיטת הקיטוע המסתגל

בשיטה זו כל עמודה מיוצגת באמצעות רצף מקטעים עוקבים ולא-חופפים. בתחילה, כל העמודה מאוחסנת במקטע יחיד, שבהדרגה מאורגן-מחדש לתוך שרשרת מקטעים בעקבות שאילתות בחירה (שליפות) שמגיעות. כל שאילתת בחירה יוצרת אפשרות לפיצול מקטעים, וההחלטה תתקבל ע"י מודל ההחלטה על הקיטוע – GD או APM. אם המודל מחליט לפצל מקטע – הוא מוחלף מיידית בשניים או שלושה תת-מקטעים (בהתאם למקרה).



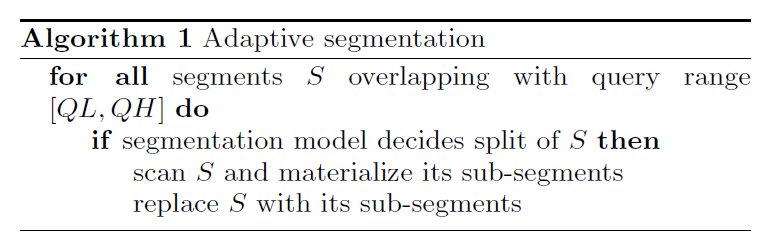
איור 3: דוגמא לתהליך "קיטוע מסתגל" עבור שלוש שאילתות רצופות.

תהליך הקיטוע מתואר באיור 3 עבור שלוש שאילתות רצופות ומודל החלטה APM. נסביר:

במצב ההתחלתי S0, העמודה מיוצגת ע"י מקטע יחיד. שאילתא Q1 גורמת מיידית לארגון מחדש לתוך שלושה מקטעים (בעקבות חוק 2 של APM). לאחר מכן, שאילתא Q2 מביאה לפיצול של התת-מקטע הראשון (שוב – חוק 2), אך לא לפיצול של התת-מקטע השני, מכיוון שהבחירה שחופפת איתו קטנה מדי (חוקים 2 ו-3 לא מתקיימים עבורו). יש לשים לב ששאילתא Q2 לא צריכה לסרוק את המקטע האחרון, שכן היא לא חופפת עם הטווח שלו. כלומר, שאילתא Q2 כבר מרוויחה מהארגון המחדש שנגרם כתוצאה משאילתא Q1. לבסוף, מכיוון ששאילתא Q3 מביאה לתחום בחירה קטן מאוד, הקיטוע מתבצע לפי הערך הממוצע של המקטע האחרון, בהתאם לחוק 3.

* 1. אלגוריתם

השלבים המרכזיים של שיטת "הקיטוע המסתגל" מופיעים באלגוריתם 1.



נתאר את מהלך האלגוריתם: עבור כל מקטע S שחופף לטווח הבחירה של השאילתא, מבוצעת פנייה למודל ההחלטה על הקיטוע. במידה והוא מחליט לפצלו - אזי מתבצעת סריקה של S, במהלכה נוצרים ומאוחסנים שניים או שלושה תת-מקטעים רלוונטיים, שמחליפים את המקטע S באמצעי האחסון.

1. שיטת השכפול מסתגל

בשיטת השכפול המסתגל, המקטעים השונים של עמודה מסוימת מאורגנים במבנה היררכי – "עץ שכפולים". מקטע S נחשב "בן" של מקטע P, אם טווח הערכים ב-P מכיל את טווח הערכים של S. שורש העץ הוא המקטע שמכיל את כל טווח הערכים של התכונה המאוחסנת בעמודה הנוכחית.

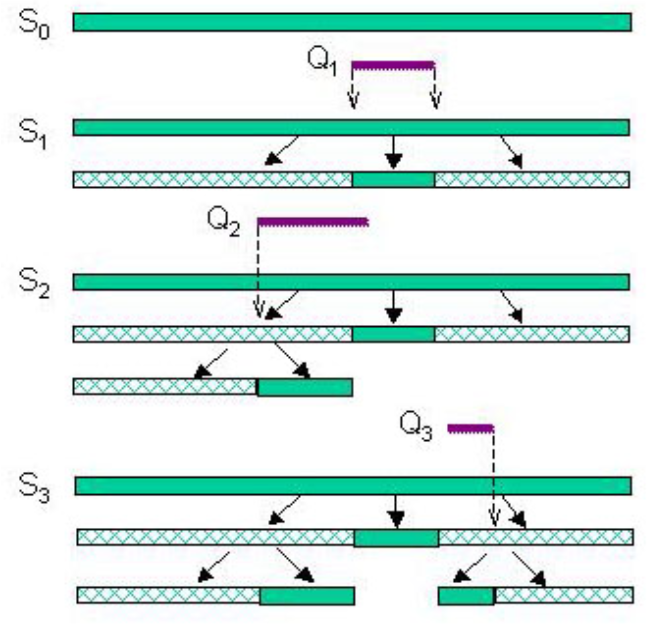
בשיטה זו עושים שימוש בשני סוגי מקטעים – מקטע ממשי (Materialized Segment) ומקטע וירטואלי (Virtual Segment). המקטעים הממשיים מכילים מידע אמיתי מטווח הערכים של העמודה (שהועתק ממנה), בעוד המקטעים הוירטואליים משמשים בעץ השכפולים בתור "משלימים" לטווחי הערכים של המקטעים הממשיים ביחס למקטע המקורי שפוצל, אך לא מכילים מידע אמיתי.

לדוג', נניח שמגיעה שאילתת בחירה עם טווח הערכים [QL,QH] שמפצלת מקטע S שמוגדר כטווח R=[SL,SH] לשני תת-תחומים: R1=[SL,QL-1], R2=[QL,SH], ונניח בשביל הפשטות שמדובר בשדה נומרי, כך שמתקיים: SL<QL<SH<QH.

הערכים המתאימים לטווח R2 כבר אוחסנו בתור תוצאת השאילתא (כי תחום הבחירה של השאילתא מכיל את כל ערכי R2), ולכן תת-מקטע ממשי שמתאים לטווח הנ"ל (S2) נוצר ומאוחסן ללא עלות נוספת (כי ערכיו מאוחסנים מראש). תת-מקטע זה הוא למעשה שכפול של ערכים מהמקטע המקורי S ולכן הוא מופיע כ-"בן" שלו בעץ השכפולים.

הטווח R1 הוא הטווח המשלים ל-R2, ביחס לטווח המקורי R. ערכי טווח R1 עדיין לא אורגנו ואוחסנו שכן השאילתא לא התעסקה איתם. לכן, עבור טווח זה נוצר תת-מקטע וירטואלי S1, המייצג את הטווח R1. תת-מקטע זה לא מכיל מידע אמיתי, וגודלו נקבע עפ"י שערוך של טווח הערכים שלו. גם תת-מקטע זה מצורף לעץ השכפולים כ-"בן" של המקטע המקורי S.

במידה ושאילתא מאוחרת יותר תעסוק בתת-מקטע S1, ייתכן והמערכת תבחר לשכפל את הערכים ולהפוך אותו לתת-מקטע ממשי. במקרה כזה – שני התת-מקטעים S1,S2 יהוו עותק מלא של המקטע המקורי S, דבר שיאפשר למחוק את המקטע המקורי S ובכך לצמצם את נפח האחסון.

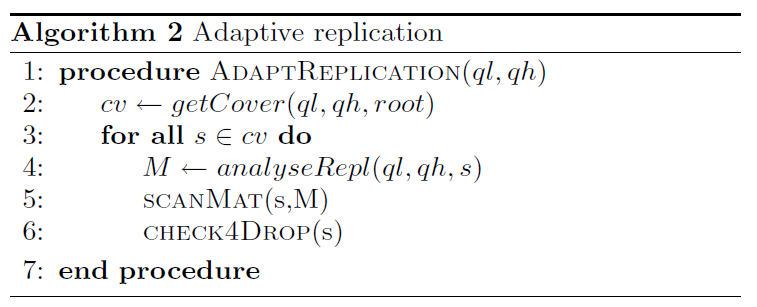


איור 4: דוגמא לתהליך "שכפול מסתגל" עבור שלוש שאילתות רצופות.

איור 4 מתאר את בניית עץ השכפולים עבור מקטע מסויים S ושלוש שאילתות רצופות. מקטע ממשי מיוצג ע"י צבע מלא ומקטע וירטואלי ע"י מילוי מפוספס. תוצאת השאילתא Q1 נשמרת כמקטע ממשי, המהווה בן של המקטע המקורי. יחד איתו נוצרים שני תת-מקטעים וירטואליים המשלימים את טווח הערכים של S. כל השלושה נשמרים כבנים ישירים של המקטע S. בהמשך מגיעות שתי שאילתות נוספות – Q2,Q3, אשר טווחי הבחירה של שתיהן חופפים לתת-מקטעים הוירטואליים. לכן, עבור כל אחת מהן יש לבצע סריקה של כל העמודה, שבמהלכה יווצר תת-מקטע ממשי המתאים לחלק הטווח החופף לתת-מקטע הוירטואלי. אופן פעולה זה סותר את שיטת ה-"קיטוע המסתגל", שבו לאחר ביצוע השאילתא הראשונה, תת-המקטעים שמתאימים לשתי השאילתות הבאות כבר היו קיימים ומאוחסנים, דבר שהיה חוסך את סריקת העמודה כולה.

* 1. אלגוריתם

השלבים המרכזיים של שיטת ה-"שכפול המסתגל" מופיעים באלגוריתם 2.



בתחילה, בהנתן טווח בחירה של שאילתא [QL,QH], המערכת מוצאת את אוסף המקטעים הממשיים המינימלי שישמש לחישוב התשובה לשאילתא (הפונקציה GetCover).

כל מקטע S באוסף נבדק ע"י מודל ההחלטה על הקיטוע, לגבי שכפולים אפשריים של חלקים ממנו (הפונקציה AnalyseReplica). במידה וההחלטה היא לייצר שכפולים (קרי, לפצל את המקטע) – נוצרים תת מקטעים חדשים בעץ, מתחת למקטע הנוכחי (גם ממשיים וגם וירטואליים). בנוסף נוצרת הרשימה M שמכילה את כל התת-מקטעים הממשיים שנוצרו, אך עדיין לא מולאו בערכים (רק נוצר מקום עבורם, אך עדיין לא שוכפלו הנתונים בפועל).

לאחר מכן, סריקה בודדת של המקטע S משמשת הן לחישוב החלק המתאים מתוצאת השליפה והן להעתקת הערכים לתת-מקטעים הממשיים החדשים שנוצרו (הפרוצדורה ScanMat).

לבסוף, האלגוריתם בודק אם כל "בניו" של המקטע S בעץ השכפולים הינם מקטעים ממשיים, ובמידה וכן – הוא מוחק את המקטע S מאמצעי האחסון ומעץ השכפולים (הפרוצדורה Check4Drop).

יש להדגיש כי באלגוריתם המוצג אין כל מגבלה על עומק עץ השכפולים, או על היקף האחסון האפשרי.

כעת נפרט לגבי כל אחד מחלקי האלגוריתם.

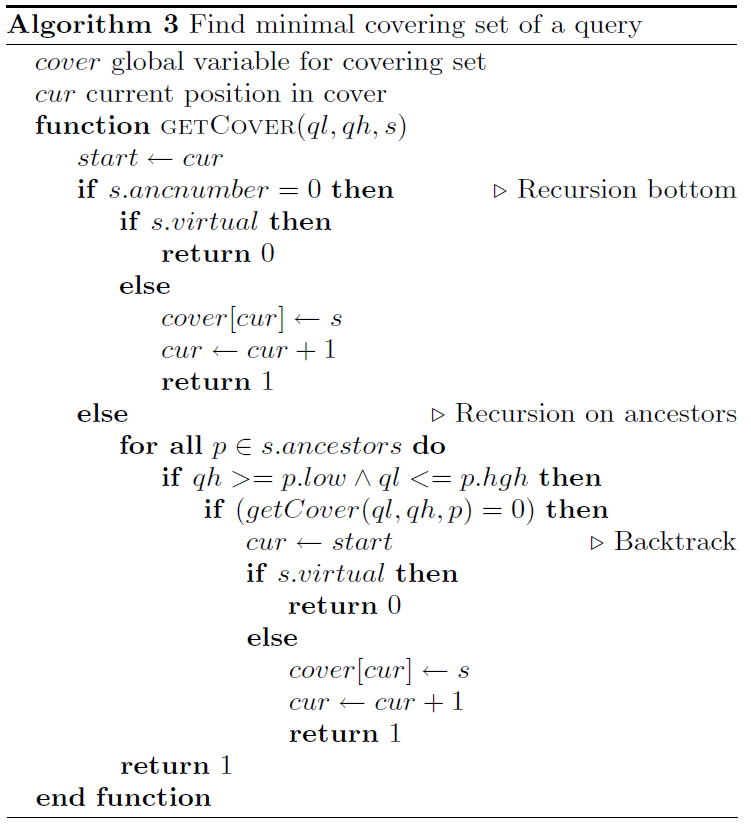
* + 1. מציאת אוסף המקטעים הממשיים המינימלי לחישוב השאילתא - GetCover

מכיוון שהמידע במערכת משוכפל חלקית, יש צורך להמנע מיצירת תוצאות זהות יותר מפעם אחת, כלומר אם תת-מקטע מסויים קיים כבר, צריך להשתמש בו ולא לייצר עותק חדש שלו. כדי לנצל את יתרונות השכפול החלקי, המערכת צריכה למצוא את אוסף המקטעים המשוכפלים האופטימלי, ובעזרתו לחשב את תוצאת השאילתא. מכיוון שמקטעים וירטואליים לא מכילים מידע, יש למצוא את האוסף המינימלי של מקטעים ממשיים שמכסים את כל טווח שאילתת הבחירה.

מבחינה פורמלית, האוסף המינימלי S= ∪Si, עבור שאילתא עם טווח בחירה [QL,QH] מוגדר באופן הבא:

1. כל המקטעים Si הם ממשיים.
2. טווח הבחירה [QL,QH] כלול באיחוד טווחי המקטעים [SLi,SHi].
3. עבור כל מקטע – לא ניתן להחליפו ב-"בניו" ללא הפרה של כלל מספר 2.
4. עבור כל מקטע – אם הוא יוסר מהאוסף, כלל מספר 2 יופר.

השלבים למציאת האוסף המינימלי מופיעים באלגוריתם 3.

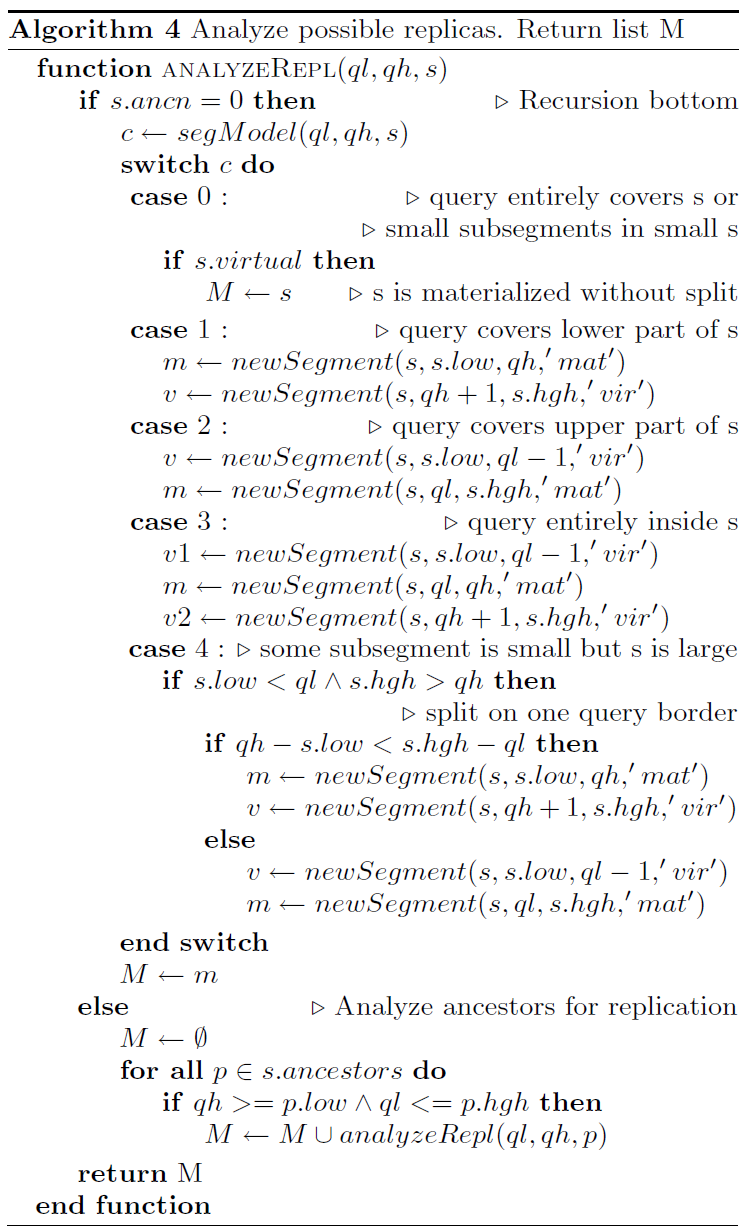


האלגוריתם הינו רקורסיבי, ובכל פעם נעשה נסיון לענות על החתך הנדרש בעזרת תת-מקטעים קטנים יותר (אלו שנמצאים למטה יותר בעץ). במידה ורמה מסוימת בעץ לא מצליחה לענות על כל החתך הנדרש, חוזרים ישירות לאב ומכניסים אותו לאוסף (כמובן רק במידה והוא לא וירטואלי. אם הוא כן – חוזרים עוד רמה אחת למעלה).

הנחה חשובה במציאת האוסף המינימלי היא שכל המקטעים נגישים במידה זהה בזיכרון, ולכן אין עדיפות למקטע מסוים על פני מקטע אחר, מעבר לנושא החפיפה עם החתך הרצוי.

* + 1. בדיקת מקטע לצורך שכפול - AnalyzeRepl

עבור כל מקטע באוסף המינימלי, האלגוריתם בודק את החפיפה בין טווח הבחירה בשאילתא וגבולות המקטע, כדי להחליט באמצעות מודל ההחלטה על יצירת שכפול. אלגוריתם 4 מהווה סקיצה של חוקי הבדיקה הזו.



פונקציה זו היא המקום בו נוצרים ומוכנסים לעץ תתי-מקטעים חדשים (הפונ' newSegment), אלו שיכילו שכפול (ממשיים) וגם אלו הוירטואליים שנועדו להשלים את טווח הערכים של המקטע המקורי. עם זאת, המקטעים הממשיים ריקים בשלב זה, ויתמלאו בערכים רק בשלב הבא – ScanMat.

הבדיקה מתחילה מהמקטע הנוכחי באוסף המינימלי, ובכל שלב יורדים רמה בעץ המתאים לו, על מנת לבדוק גם את בניו. כאשר מגיעים לתת-מקטע שאין לו בנים, מבוצעת קריאה למודל ההחלטה, כדי להבין מהו הקשר בין טווח הערכים הרצוי והתת-מקטע הנוכחי.

בהתאם לרעיון המרכזי של יצירת שכפולים רק עבור חלקים שהשאילתות מתעניינות בהם, במידה וטווח השאילתא חופף לחלק התחתון של התת-מקטע הנבדק (מקרה 1) או לחלק העליון של התת-מקטע (מקרה 2) או לחלק האמצעי שלו (מקרה 3) – נוצרים ומתווספים לעץ תת-מקטעים ממשיים שאמורים להכיל את השכפולים. גם תתי-המקטעים הוירטואליים שמשלימים את הטווח נוצרים בשלב זה. בנוסף, תתי-המקטעים הממשיים החדשים מוכנסים לרשימה M, כדי שימולאו בערכים בשלב הבא.

קיימים שני מקרי קצה:

* מקרה 0 מתאים למצב שטווח השאילתא מכסה את כל המקטע הנבדק, או שהוא קטן מדי והשכפולים שיווצרו יהיו קטנים מהגבול המותר. במקרה כזה, כל שיש לעשות הוא להפוך את המקטע לממשי, במידה והוא וירטואלי, ולא להביא לשכפולים נוספים. במידה והוא כבר ממשי – אין יותר מה לעשות, ולא משכפלים חלקים ממנו (כי הם יהיו קטנים מדי). מצב זה מתאים למעשה למקרה 1 במודל APM.
* מקרה 4 מתאים למצב שהמקטע הנבדק גדול יחסית, אך טווח השאילתא מביא ליצירת שכפול של חלק קטן מאוד. במצב זה, במקום לשכפל את החלק הקטן, יוצרים שכפול עד לאחד מגבולות טווח השאילתא, כדי לעמוד בגודל המינימלי המותר (זהו למעשה מקרה 3 במודל APM).
  + 1. בניית מקטעים ממשיים וחישוב תוצאת שאילתא - ScanMat

לאחר שנוצרו תת-מקטעים ממשיים ווירטואליים לטובת השכפולים הנדרשים, יש צורך להעתיק את הערכים המתאימים לתתי-המקטע ממשיים. כדי לבצע זאת, מבוצע מעבר על המקטע המקורי, וערכיו מועתקים למקטעים הממשיים המתאימים.

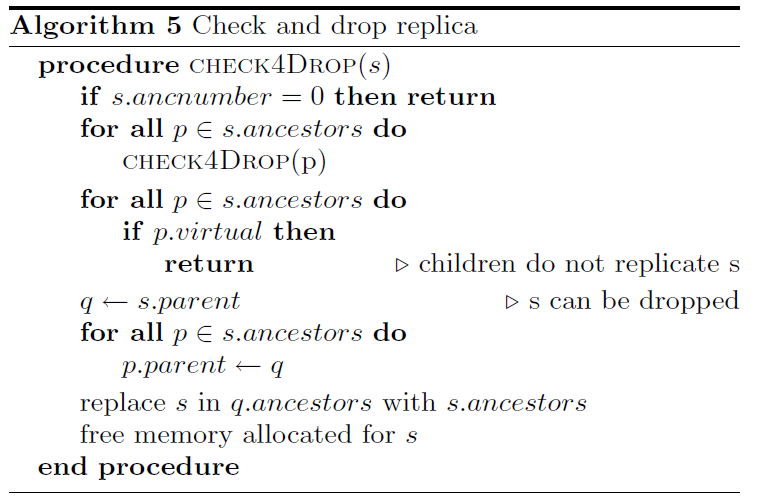
כמו כן, בשלב זה תוצאות השאילתא מחושבות עבור הטווח שכלול במקטע הנוכחי.

* + 1. הסרת מקטע ששוכפל במלואו – Check4Drop

ככל שעובר הזמן, מספר העותקים גדל, והמערכת צריכה לדאוג שלא להשתמש ביותר מדי משאבי אחסון. דבר זה מושג ע"י בדיקה וחיפוש מתמיד אחר עותקים שניתן להסירם. במידה וכל ה-"בנים" הישירים של מקטע מסויים בעץ השכפולים הם "מקטעים ממשיים", הרי שניתן להסיר בבטחה את המקטע המדובר מהעץ. בנוסף, בניו של המקטע מקושרים ישירות לאביו.

במידה והמקטע שהוסר גם הוא ממשי (ולא וירטואלי) – הרי שמדובר בשחרור של משאבי אחסון.

שלבי הביצוע של הבדיקה וההסרה מפורטים באלגוריתם 5.



האלגוריתם הינו רקורסיבי ובודק עבור כל מקטע אם ניתן להסיר כל אחד מתתי-המקטעים שלו. לאחר מכן הוא בודק אם ניתן להסיר את המקטע עצמו, ע"י בדיקה שכל "בניו" אינם וירטואליים. במידה וניתן להסיר – הוא משחרר את שטח האחסון שלו, ומעדכן את המצביעים השונים בעץ השכפולים.

1. הערכה ניסויית

ההערכה הניסויית שבוצעה עבור הפרוייקט כוללת שני חלקים:

* סימולציה מעמיקה של שתי טכניקות הארגון העצמי
* ניסוי בסביבת MonetDB עבור "קיטוע מסתגל", הכולל עומס עבודה אמיתי, על 100GB של מידע. המידע הגיע מפרוייקט בשם SkyServer.
  1. סימולציה

במסגרת העבודה, בוצעה סימולציה של אלגוריתמי הליבה של MonetDB, של ניהולו מעל סביבה מוגבלת בזיכרון, ושל התנהגותו בפעולות קריאה וכתיבה כאשר מידע מועבר לאחסון משני (לדוג' דיסק קשיח). הסימולטור שנוצר מודע למעשה לארכיטקטורת ה-MonetDB, וככזה, הוא מהווה בסיס טוב לבחינת הטכניקות שמוצגות בעבודה. נקודה זו אומתה לפני שהחלה העבודה הניסויית.

התוצאות שיפורטו בהמשך מבוססות על עמודה המכילה 100K ערכים, מתוך מרחב של 1M ערכים מספריים אפשריים. נבדקו כך שתי השיטות – "קיטוע מסתגל" ו-"שכפול מסתגל", יחד עם שני מודלי ההחלטה על הקיטוע – GD ו-APM (גבולות ה-APM נקבעו ל: Mmin=3KB, Mmax=12KB).

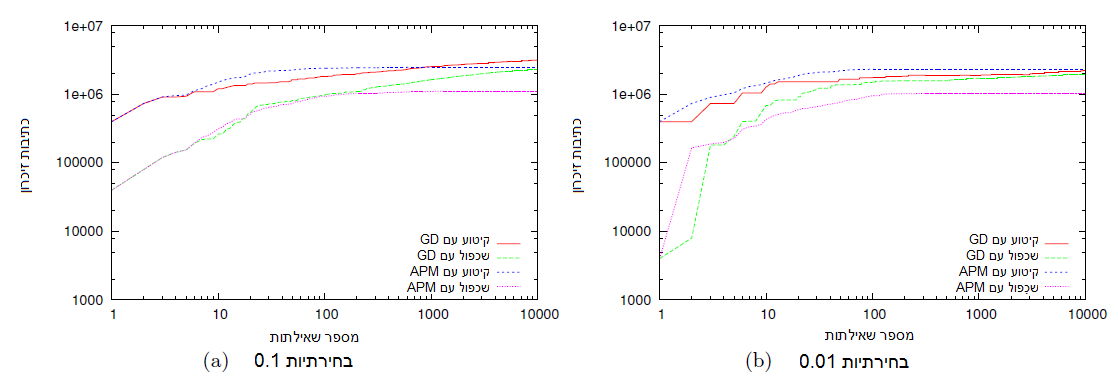
עומס העבודה בניסוי מכיל שאילתות בחירה בעלות טווח בחירה של 10K ערכים.

על מנת לבחון התנהגויות שונות בעומסים שונים, נבחרו שתי רמות סלקטיביות (היחס בין מספר הערכים העונים לתנאי השאילתה לבין טווח הערכים המקורי בעמודה) – 0.1 ו- 0.01, ובנוסף שני אופני התפלגות של הערכים הנבדקים בשאילתות השונות – התפלגות אחידה והתפלגות מסוג Zipf (התפלגות הנוטה לאיזור מסויים בטווח).

בהמשך הפרק נציג את תוצאות הסימולציה, ונעלה תובנות הקשורות לתקורה המתקבלת ביחס ליתרונות של הארגון העצמי, ובנוסף נבחן את גודל עץ השכפולים בהתאם למודל ההחלטה המופעל.

* + 1. תקורת הארגון העצמי

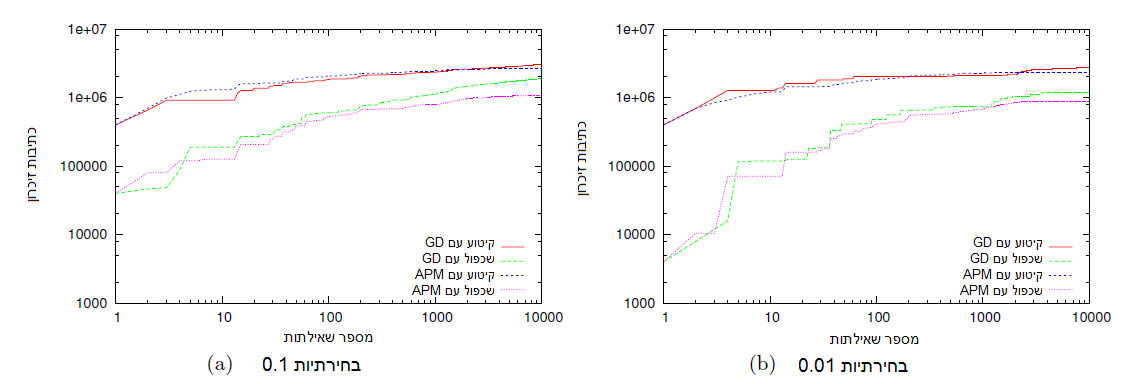
על מנת להעריך את תקורת הארגון-מחדש, נמדד המספר הצובר של כתיבות לזיכרון הנובעות מאחסון מקטעים חדשים המכילים את תוצאות השאילתות. תוצאות המדידה מופיעות באיורים 5 ו-6, עבור התפלגות אחידה והתפלגות Zipf, בהתאמה.



סלקטיביות 0.1

סלקטיביות 0.01

איור 5: כמות צוברת של כתיבות זיכרון בעקבות יצירת ואחסון מקטעים. התפלגות אחידה.



סלקטיביות 0.01

סלקטיביות 0.1

איור 6: כמות צוברת של כתיבות זיכרון בעקבות יצירת ואחסון מקטעים. התפלגות Zipf.

עבור כל שילוב של רמת סלקטיביות והתפלגות, שכפול מסתגל מצריך פחות כתיבות מאשר הקיטוע המסתגל. עובדה זו מאששת את הציפיות, שכן שיטת השכפול יוצרת ומאחסנת מקטעים באופן "עצל", רק לאחר ששאילתא מסויימת פונה אליו.

באופן כללי, מודל ה-APM מפסיק לארגן מחדש את העמודה לאחר מספר התחלתי של שאילתות. המצב מתקבל בהתפלגות אחידה לאחר כ-100 שאילתות, בעוד עבור התפלגות zipf ניתן לראות ארגונים-מחדש גם לאחר 3000 עמודות, מכיוון שניגשים בפעם הראשונה לאיזורים מסויימים במרחב הערכים של העמודה רק מאוחר יותר. דבר זה מוביל לארגונים-מחדש מאוחרים יחסית.

בסתירה לכך, מודל ה-GD ממשיך לבצע ארגונים-מחדש עם הסתברות פוחתת, מכיוון שבמודל זה אין גבולות ברורים לגודלי המקטעים (ניתן לראות זאת בחלק האחרון של הגרפים השונים).

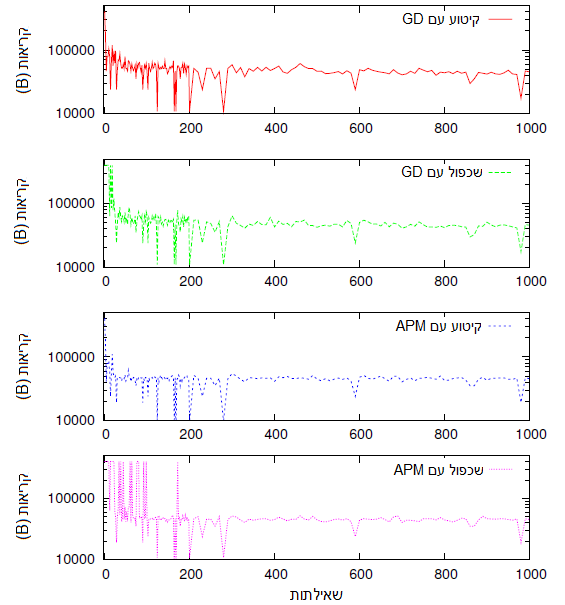
אם נתבונן במצבי הגרפים בשאילתות הראשונות, נסיק שמודל ה-GD פחות אגרסיבי בשיטת השכפול, ולכן ניתן להעדיף אותו אם רוצים להפחית תקורה בתחילת העבודה.

רמת הסלקטיביות בעומס השאילתות משפיעה על החלטות הקיטוע. בעוד שיטת הקיטוע המסתגל מארגנת מחדש מקטע שלם ללא תלות בגודל תוצאת הבחירה בשאילתא (ששווה לגודל התת-מקטע שיווצר), התקורה בשיטת השכפול המסתגל מושפעת ישירות מגורם הסלקטיביות (שמשפיעה ישירות על גודל התת-מקטע שיווצר), כפי שניתן לראות עבור השאילתות הראשונות בגרפים. דבר זה הגיוני, שכן עבור רמת סלקטיביות נמוכה יותר, השכפול הוא של חלק קטן יותר, דבר המוביל למספר נמוך יותר של כתיבות זיכרון, בעוד שבשיטת הקיטוע המסתגל כל התת-מקטעים של המקטע השלם ייכתבו מחדש. מסקנה זו רומזת לכך ששיטת השכפול המסתגל מצריכה תשומת לב ושמירה כנגד שכפולים של חלקים גדולים מדי.

* + 1. יתרונות הארגון העצמי

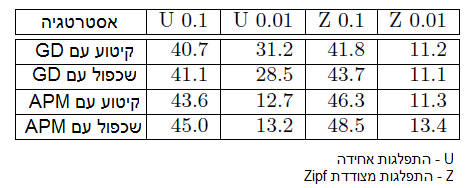
כעת, נחקור כיצד השיטות השונות משיגות את המטרה העיקרית של הקיטוע – שיפור הביצועים ע"י הגבלת סריקות המידע רק לחלקים מהעמודה הרלוונטיים לשאילתא.

איור 7 מראה את מספר קריאות הזיכרון (הכוונה בקריאת זיכרון היא לקריאה של בית זיכרון בודד) במהלך 1000 השאילתות הראשונות עבור התפלגות ערכים אחידה. כצפוי, מספר הקריאות יורד מהר מאוד עם שיטת הקיטוע המסתגל, עבור שני מודלי ההחלטה. אם נתבונן בתחילת העקומות של שיטת השכפול המסתגל (ובמיוחד ב-APM), נראה מספר עליות חדות שמתאימות לסריקות מלאות של העמודה. עליות אלו נובעות משאילתות שמבצעות בחירה מתוך איזורים שלא נגעו בהם קודם ושלא כוסו ע"י עותקים חלקיים קטנים יותר (לדוג' כפי שקרה עבור שאילתות Q2,Q3 בפרק 7). בכל המקרים, מספר הקריאות פוחת ומתייצב ככל שעומס השאילתות ממשיך, שכן רוב הקיטועים בוצעו.



איור 7: מספר קריאות הזיכרון עבור 1000 השאילתות הראשונות. התפלגות אחידה, סלקטיביות 0.1.

טבלה 1 מראה את המספר הממוצע של קריאות זיכרון לשאילתא בודדת (הכוונה בקריאת זיכרון היא לקריאה של בית זיכרון בודד), עבור כל מהלך הסימולציה של 10K שאילתות. עבור עומס עבודה עם רמת סלקטיביות של 0.1, שבו תוצאות הבחירה מיוצגות ע"י מקטעים גדולים יחסית, מספר קריאות הזיכרון מתכנס למספר המינימלי של 40K, עבור כל השיטות וההתפלגויות, כאשר הערכים עבור שיטת השכפול המסתגל גבוהים מעט יותר משיטת הקיטוע המסתגל (בהתאם לנאמר קודם).



טבלה 1: מספרי קריאות זיכרון ממוצעים לשאילתא בודדת ב-KB עבור 10K שאילתות.

כאשר רמת הסלקטיביות היא 0.01, התת-מקטעים שמייצגים תוצאת שליפה הם קטנים יחסית למקרה של 0.1.

בהתאם לכך, מספר הקריאות במודל APM ורמת סלקטיביות של 0.01, מתכנס ל: 11-13KB, ולא מגיע לגבול המינימלי שנקבע ל-3KB. הסיבה לכך היא ששליפות קטנות מדי (מה שקורה עבור 0.01) מובילות לקיטוע רק אם מתקיים SizeS>Mmax, שנקבע ל- 12KB. לכן לא מתרחשים הרבה קיטועים כאלה, ורוב המקטעים נשארים סביב ה-12KB. מכיוון שנקראים מקטעים שלמים, מספר הקריאות לא יכול לרדת מגודלי המקטעים, ולכן הוא מתכנס ל11-13KB.

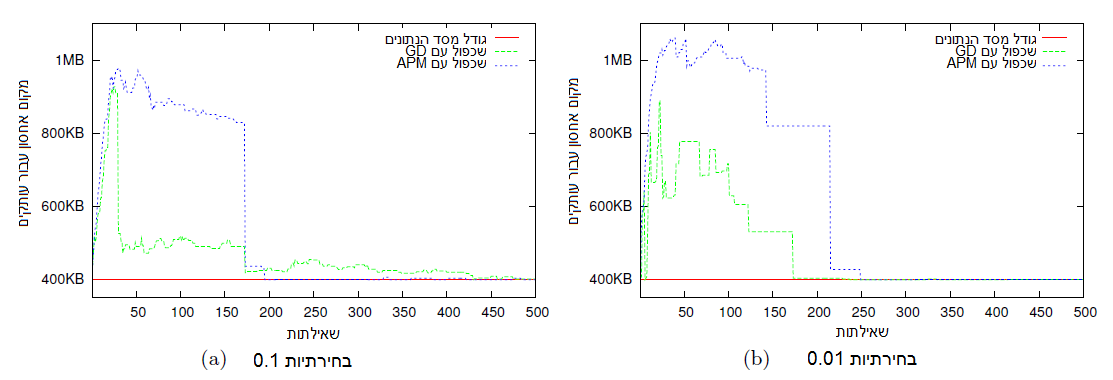
עבור מודל GD ורמת סלקטיביות של 0.01, ניתן לראות שנוצרים מקטעים גדולים יחסית עבור התפלגות אחידה. דבר זה קורה בגלל אופן הפעולה של GD: מכיוון שרמת סלקטיביות נמוכה מתאימה לערכי x נמוכים במודל GD, הרי שההסתברות לקיטוע היא נמוכה יותר, ולכן ארגון-מחדש קורה לעיתים נדירות יותר, והמקטעים נשארים גדולים יחסית.

* + 1. גודל עץ השכפולים

מכיוון ששכפול מסתגל דורש יותר מקום אחסון, נעשה שימוש בסימולטור גם לצורך בחינת הפרמטרים של עץ השכפולים בהתאם למקרים השונים.

עבור התפלגות אחידה, עץ השכפולים מצריך מקום אחסון נוסף של פי 1.5 מגודל העמודה, אך הצורך פוחת משמעותית לאחר 250 השאילתות הראשונות (כפי שניתן לראות באיור 8). הירידות הגדולות ביותר בעקומת האחסון מתאימות למצבים שבהם מקטע התחלתי שמכיל את כל העמודה שוכפל באופן מלא ע"י בנים ממשיים שלו, וניתן להסירו מאמצעי האחסון. ככל שיותר ויותר מקרים כאלה מתרחשים, עץ השכפולים הופך למבנה דומה מאוד לרשימת המקטעים שנוצרת בעקבות שיטת הקיטוע המסתגל, עם שכפולים קטנים ומקריים.

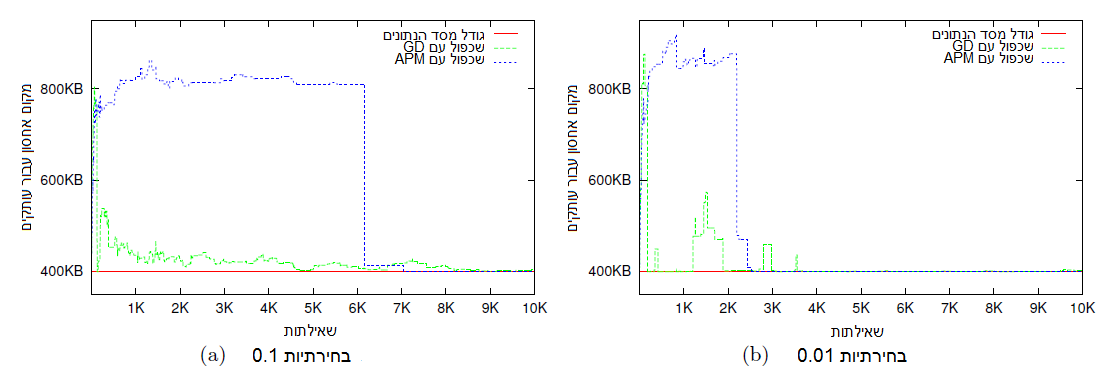
תהליך זה מתרחש גם עם התפלגות zipf של השאילתות (ראה איור 9), אבל לוקח הרבה יותר זמן (בין 3-6K שאילתות), עד שהשאילתות פונות לכל האיזורים של מרחב הערכים, ומובילות לארגון מחדש של כל העמודה.



סלקטיביות 0.01

סלקטיביות 0.1

איור 8: מקום אחסון עבור עץ השכפולים. התפלגות אחידה.



סלקטיביות 0.01

סלקטיביות 0.1

איור 9: מקום אחסון עבור עץ השכפולים. התפלגות Zipf.

הבחנה נוספת היא שבכל אחד מהמקרים, הצורך במקום אחסון עודף במודל GD פוחת מהר יותר לעומת מודל APM. הסיבה המרכזית לכך, היא ששני המודלים השונים מטפלים באופן שונה במקטעים גדולים יחסית עם בחירות שמהוות תתי-מקטעים קטנים יחסית:

מודל ה-APM יפצל מקטע ממשי S רק אם SizeS>Mmax, וכתוצאה יווצר תת-מקטע ממשי שמכיל את תוצאת הבחירה. לכן, APM מקצה מקום אחסון נוסף עבור השכפול, אבל צריך לחכות עד ששאילתא אחרת תיגע בתתי-המקטע הוירטואליים המשלימים, לפני שיתקבל עותק מלא של המקטע המקורי S, וניתן יהיה להסיר אותו ממקום האחסון.

במודל GD, מכיוון שתוצאת הבחירה קטנה יחסית, הפרמטר x יהיה קטן יותר, ולכן בהסתברות גבוהה המקטע S לא יפוצל, ולא יידרש מקום אחסון נוסף עבור שכפול.

באופן דומה, אם המקטע S הוא וירטואלי, אזי החלטת המודל GD שלא לפצלו גורמת להפיכתו לממשי באופן מיידי, דבר שיכול להוביל להסרת מקטע האב שלו (במידה ו-S היה הבן הוירטואלי היחיד של P), ולשחרור מקום אחסון. במקרה כזה, מודל APM שוב יחליט על פיצול, וישכפל את התת-מקטע של S ואז יחכה לשאילתא אחרת שתוביל למימוש ואחסון תתי-המקטעים המשלימים, כדי לשחרר את מקום האחסון שתופס מקטע האב P.

* 1. ניסוי בסביבה אמיתית

על מנת לבסס את הסימולציה, יש צורך בניסויים בסביבה אמיתית. לכן, ע"י שימוש ביכולת ההרחבה של ה-MonetDB כפי שתוארה בפרק 2, בוצעה הרחבה של מערכת מסד הנתונים באבטיפוס של שיטת הקיטוע המסתגל, ובוצעו הרצות ניסוי על גבי מערכת מורחבת זו. בפרק זה יפורטו התוצאות שהתקבלו באמצעות אלגוריתם "קיטוע מסתגל" על מידע ועומס עבודה אמיתי כפי שנלקחו מפרוייקט בשם SkyServer. לא נעשו הרצות ניסוי על אלגוריתם "שכפול מסתגל".

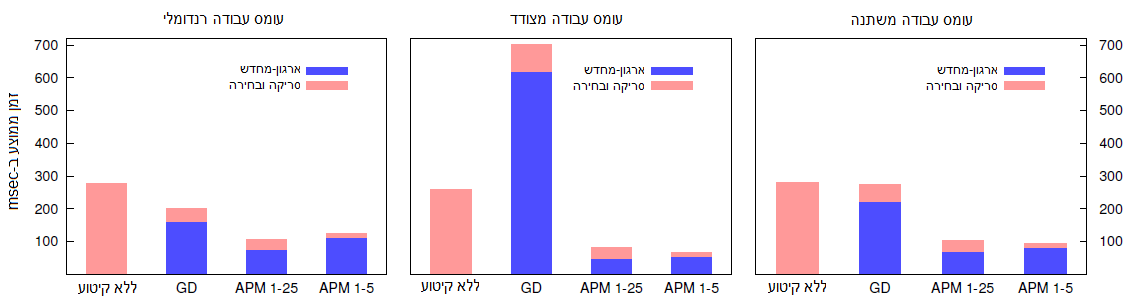
הפלטפורמה ששימשה לניסוי היא מחשב PC עם מעבד כפול-ליבה מסוג AMD Opteron(tm) 270 2GHz, וזיכרון בגודל 8GB. לטובת הניסוי נלקח מדגם בגודל 100GB של נתונים ממסד הנתונים SDSS-4. כמות כזו עדיין מתאימה למחשב ביתי פשוט, אבל כבר בהחלט הופכת את מסד הנתונים להיות תלוי מאוד בזמני גישה לדיסק הקשיח עבור רוב השאילתות, אפילו כאשר הוא מבוסס עמודות.

כדי לייצר "עומס עבודה" של שאילתות, סוננו רק השאילתות שחופפות עם ה- 100GB של המידע, מתוך לוג שאילתות של חודש אחד מפרוייקט SkyServer. העמודה שהתמקדו בה היא "עלייה ימנית" (Right Ascension - RA), שהיא טיפוס מידע אמיתי שנעשה בו שימוש בשאילתות חיפוש מרחביות. נלקחו שלושה עומסי עבודה שונים, כל אחד עם 200 שאילתות:

* RANDOM - אקראי- בחירות רנדומליות של אחת מכל 300 שאילתות, כך שכל מרחב הערכים יכוסה באופן אחיד.
* SKEW - מצודד - 200 שאילתות עוקבות מהלוג שניגשות לשני חלקים מאוד מוגבלים של מרחב הערכים.
* CHANGING - משתנה - 4 חלקים שונים של 50 שאילתות עוקבות, כאשר כל חלק פונה לאיזור אחר במרחב הערכים (טווח העניין משתנה לאחר כל 50 שאילתות).

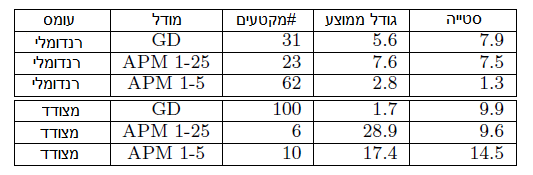
בחלק זה של ההערכה, נעשה שימוש הן במודל ההחלטה GD והן במודל ההחלטה APM, כאשר האחרון נבדק בשתי גרסאות שונות: בראשונה Mmax=5MB, בשנייה Mmax=25MB ובנוסף Mmin=1MB עבור שתיהן.

הציפייה הייתה שהשימוש בקיטוע מסתגל יוביל לזמני עיבוד שאילתות טובים יותר מאשר במסד-נתונים רגיל, לאחר שהתקורה מהשאילתות הראשונות תופחת בעקבות שליפות מאוחרות ומהירות יותר. בנוסף היה צפוי ששיטות הארגון העצמי יתאימו טוב יותר לעומס העבודה המצודד.



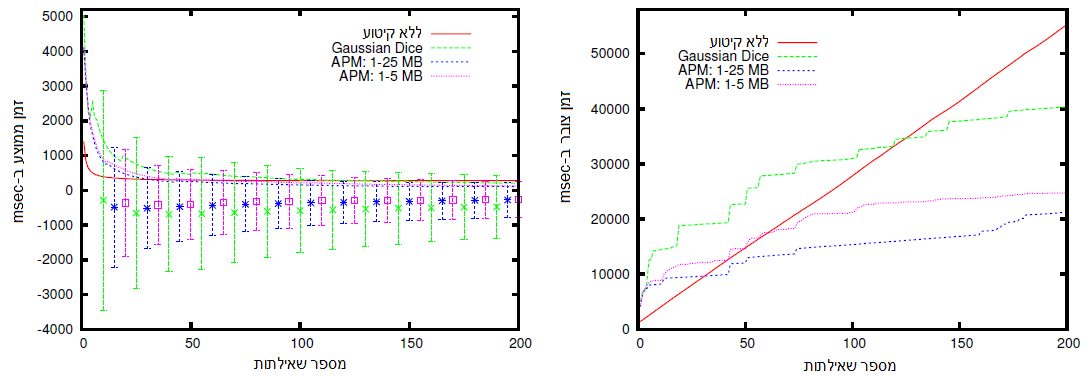
איור 10: זמנים ממוצעים לארגון-מחדש אל מול שליפת הנתונים (סריקה ובחירה).

איור 10 מראה את הזמנים הממוצעים שלקחו תהליכי השליפה (סריקה ובחירת הנתונים מהמקטעים השונים) אל מול תהליכי הארגון-מחדש (החלטה לגבי קיטוע וביצוע הקיטוע עצמו), לאחר 200 השאילתות הראשונות. עבור שלושת עומסי העבודה, ניתן לראות שתקורת הארגון-מחדש עבור APM קטנה מהמצב עם GD, מכיוון ש-APM הרבה יותר "שמרני" בפיצול מקטעים קטנים. באופן דומה ניתן לראות שהתקורה עבור APM1-5 (הגרסה הראשונה) גבוהה יותר מאשר עבור APM1-25, שכן בגרסה השניה לא יפוצל מקטע שקטן מ-25MB, במידה ונבחר חלק קטן מאוד (פחות מ-1MB) ממנו.



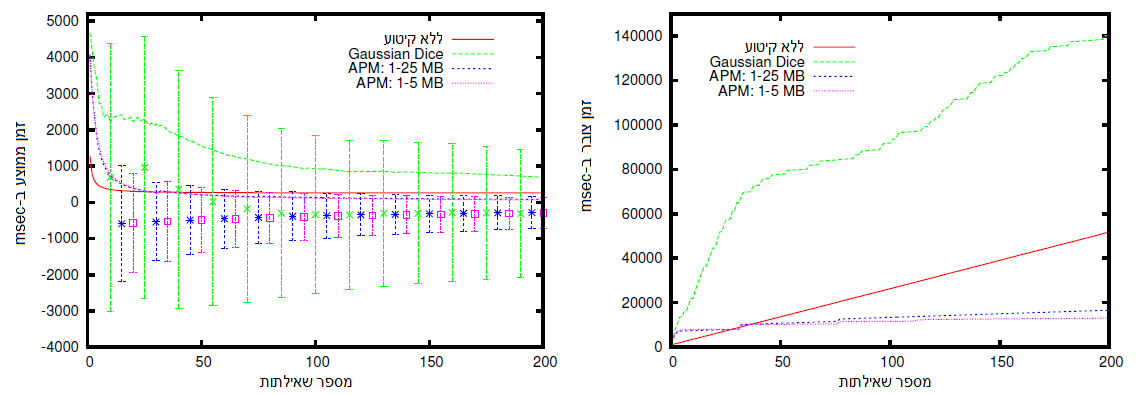
טבלה 2: סטטיסטיקות על המקטעים במקרים השונים.

עקב הגבול העליון הקטן יותר, מודל APM1-5 מביא ליצירת מקטעים קטנים יותר מאשר מודל APM1-25, כפי שניתן לראות בטבלה 2. מקטעים קטנים יותר מביאים לזמני בחירה קצרים יותר, שכן הסריקה לוקחת פחות זמן (כי הטווח הנסרק קרוב יותר לטווח הבחירה). ניתן לראות זאת באיור 10, בעמודות המתאימות לשני גרסאות ה-APM.



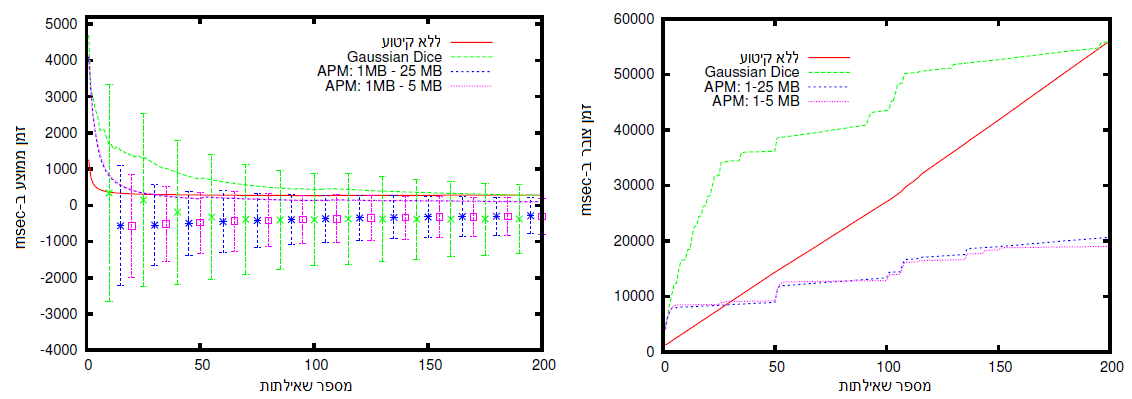
איורים 11 ו-12: זמן צובר של ביצוע השאילתות וזמן ממוצע לביצוע שאילתא, בהתאמה, עבור עומס רנדומלי.

איורים 11 ו-12 מראים את זמן ביצוע השאילות הצובר ואת הזמן הממוצע לביצוע שאילתא, בהתאמה, עבור קיטוע מסתגל והמודלים השונים לעומת מסד נתונים ללא קיטוע. האיורים התקבלו מעומס העבודה הרנדומלי. כפי שניתן לראות, השאילתות הראשונות בשלושת אסטרטגיות הקיטוע איטיות יותר ממסד נתונים רגיל, עקב התקורה התחלתית של הארגון-מחדש, אך לאחר מספר קטן יחסית של שאילתות מתקבלים זמנים הרבה יותר טובים. הראשון להשיג זאת הוא מודל APM1-25, שעוקף את המסד הרגיל לאחר כ-30 שאילתות (תוצאה זו תואמת לנאמר בפסקה הקודמת ולטבלה 2).



איורים 13 ו-14: זמן צובר של ביצוע השאילתות וזמן ממוצע לביצוע שאילתא, בהתאמה, עבור עומס מצודד.

איורים 13 ו-14 מקבילים לשני האיורים הקודמים, אולם כעת עבור עומס העבודה המצודד. התקורה הכוללת שמתקבלת במודלי ה-APM קטנה יותר מזו של עומס העבודה הרנדומלי, מכיוון שכעת הארגון-מחדש משפיע על איזור מאוד מוגבל של מרחב הערכים. עם זאת, עומס העבודה המצודד הוא המקרה הגרוע ביותר עבור מודל GD. בעומס המצודד, השאילתות כמעט ולא שונות בטווח הבחירה שלהן, דבר שמוביל ליצירת תתי מקטעים קטנים מאוד. כתוצאה מכך, 80% מהמקטעים מכילים פחות מ-1000 ערכים, ולכן יקר יותר לארגן אותם מחדש, ובנוסף נדרש לחבר מספר חלקים קטנים יחד עבור שאילתות עוקבות.



איורים 15 ו-16: זמן צובר של ביצוע השאילתות וזמן ממוצע לביצוע שאילתא, בהתאמה, עבור עומס משתנה.

איורים 15 ו-16 גם הם מקבילים לשניים הקודמים, כאשר כעת מדובר בעומס העבודה המשתנה. איורים אלה מדגימים כיצד שינוי טווח העניין של השאילתות מוביל לארגון-מחדש של איזורים מהעמודה שלא נגעו בהם קודם. התוצאה של פעולה זו היא העלייה בתקורה לאחר 50 ו-100 שאילתות (נקודות שינוי טווח העניין), כאשר מיד לאחר מכן היא קטנה בחזרה.

1. סיכום

תחום מסדי הנתונים מבוססי אחסון העמודות הינו תחום רחב מאוד, הנותן מענה למערכות תוכנה רבות העוסקות בניתוח מידע רב, עם עומסי-עבודה משתנים. תחום זה נחקר רבות במוסדות שונים, וביניהם במכון המחקר הלאומי בתחומי מתמטיקה ומדעי המחשב בהולנד. בעבודה זו העמקנו בשני נושאים מעולם אחסון העמודות, כפי שנלמדו במכון הנ"ל – הצגנו מערכת לניהול מסדי נתונים בשם MonetDB, ותיארנו שיטות לארגון עצמי של המידע המאוחסן במסדי נתונים מסוג זה.

מערכת ה-MonetDB מהווה כבר היום מערכת ניהול מסדי נתונים מרכזית בעולם אחסון העמודות, דבר שבא לידי ביטוי בכמות ההורדות שלה, שהולכת וגדלה. למדנו על המבנה והארכיטקטורה של המערכת, ושמנו דגש על ההבדלים בינה לבין מערכות קלאסיות לניהול מערכות בסיסי נתונים. בזכות הבדלים אלו, מערכת ה-MonetDB משגשגת בתחומי מחסני הנתונים והאנליזות על מידע רב.

צוות המחקר שפיתח את המערכת ממשיך לחקור בנושא, כאשר הדגש הוא בשני נושאים מרכזיים: עיבוד מבוזר ומקבילי של שאילתות, ועיבוד מסתגל של שאילתות – עיבוד שיתבצע עפ"י ההקשר של השאילתא עצמה ולא עפ"י חוזה מוסכם מראש.

בנוסף להצגת ה-MonetDB, הצגנו את הרעיון של "ארגון-עצמי" של המידע המאוחסן במסדי נתונים מבוססי אחסון עמודות. רעיון זה תקף למסדי-נתונים כאלו באופן כללי ולאו דווקא ל-MonetDB. הצגנו שתי שיטות מרכזיות המשתלבות בהרצת השאילתות – "קיטוע מסתגל" ו-שכפול מסתגל", שיטות שהוכחו כמשפרות ביצועים ומפשטות את הניהול ותחזוקה. הסימולציה והרצות הניסוי מעל מערכת ה-MonetDB אישרו את הציפיות לביצועים טובים יותר, בעקבות הפנייה היעילה והפרטנית לחלקים המתאימים מהעמודות.

שתי השיטות מוכיחות את יעילותן במקרים שונים: שיטת הקיטוע המסתגל משתמשת במקום אחסון מינימלי כדי לנהל את הקיטוע, אבל התקורה בתחילת העבודה הינה רבה. בשונה ממנה, שיטת השכפול המסתגל בוחרת להשתמש בשטחי אחסון נוספים על מנת להוריד את התקורה שבקיטוע.

באופן דומה, השימוש במודלי ההחלטה גם תלוי במקרה הספציפי: מודל הקיטוע APM ייבחר כאשר יש עדיפות להפחתה מתמשכת בתקורת הארגון העצמי, עם ההתקדמות בעבודה. בשונה ממנו, מודל הקיטוע GD ייבחר אם יש חשיבות להפחתה של התקורה בתחילת עבודת המערכת או של מקום האחסון העודף הדרוש לשכפול המסתגל.

עפ"י הכתוב במאמר, המחקר בנושאי שיטות הארגון העצמי הללו לא הסתיים, ונדרשת עבודה נוספת בתחומים שונים, ביניהם: התאמת אלגוריתם השכפול המסתגל לסביבה המוגבלת מבחינת המשאבים, קביעה אוטומטית של הפרמטרים למודל ה-APM ויצירת שיטות משלימות למודל ה-GD שיאחדו תתי-מקטעים קטנים מדי. תחום נוסף שיש לבחון הוא שילוב תהליכי הקיטוע במערכות בסיסי נתונים מבוזרות.

למרות העבודה הנדרשת הזו, נושא הארגון-העצמי מספק כבר היום תועלת, בתחום שרובו מבוסס על הנחיות אנושיות וקביעה סטטית של חוקים.

באופן אישי, תהליך כתיבת העבודה חידש לי רבות, מכיוון שנחשפתי בפעם הראשונה לנושא אחסון העמודות. רוב השימוש הסטנדרטי הוא בבסיסי נתונים קלאסיים, מבוססי אחסון שורות, והיה מעניין ללמוד על שיטה שונה לחלוטין לארגון המידע עצמו, המתאימה לתחומים חדשים ומתפתחים בעולם התוכנה, תחומים שכל אדם שעוסק במדעי המחשב או בפיתוח תוכנה יכול ועתיד להחשף אליהם.

1. ביבליוגרפיה
2. Idreos, S., Groffen, F., Nes, N., Manegold, S., Mullender, K. S., Kersten, M.

MonetDB: Two decades of research in column-oriented database architectures.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 35,

pp. 40-45, 2012.

1. Milena Ivanova, Martin L. Kersten, and Niels Nes.

Self-organizing strategies for a column-store database.

Proceedings of the 11th international conference on Extending database technology: Advances in database technology (EDBT '08). ACM, pp. 157-168., 2008.