האוניברסיטה הפתוחה

המחלקה למתמטיקה ולמדעי המחשב

Data Lake יעיל ומהיר עבור נתונים על מגפת COVID-19

דוח פרויקט מתקדם זה הוגש כחלק מהדרישות לקבלת תואר

"מוסמך למדעים" .M.Sc במדעי המחשב

באוניברסיטה הפתוחה

המחלקה למתמטיקה ולמדעי המחשב

על-ידי

גיל ערן

ת.ז 308047828

העבודה הוכנה בהדרכתו של פרופ' אהוד גודס

ינואר 2023

תוכן עניינים

1 רשימות

1.1 רשימת איורים

1.2 רשימת טבלאות

2 מבוא

הפרויקט מממש Data Lake בעל שני סוגי אינדקסים המאפשרים שליפה מהירה על תוכן הקבצים ועל הmetadata המתאר את הקבצים. מטרת הפרויקט היא מימוש אינדקס בהתאם למאמר [10] וביצוע שיפורים נוספים המוצעים לטובת הוכחת היתכנות ופרקטיות של מימוש השיפורים האלו. בנוסף, הפרויקט מציע גם אינדקסים הסתברותיים המבוססים על המאמרים הבאים [2] [3] על הmetadata של הקבצים בצורה חסכונית במקום ובביצועים, ובכך מתאפשרת השוואה בין שתי הגישות.

בנוסף לאינדקסים המתוארים מעלה מימשתי API מעל הdate lake המאפשר את הפעולות הבאות:

- העלאה של קובץ לdata lake והכנסה שלו לכל האינדקסים
- תשאול הdata lake עבור ערכים מסוימים וקבלת הקבצים המכילים אותם
- תשאול הdata lake עבור סטטיסטיקות מקורבות על סוגי הקבצים וה-metadata שלהם

הפרויקט נכתב ב-C# ומתבסס על .NET Framework, Java. הפרויקט מתבסס בנוסף על מספר טכנולוגיות מבוססות קוד-פתוח לטובת יצירת ארכיטקטורה יציבה ואמינה, ביניהן Kafka, Redis, Tika. בנוסף, חשוב לציין שהפרויקט בכללותו מתבסס על שימוש בשירות S3 של חברת Amazon. תלות זו ניתנת להחלפה בהתאם לצורך

את הקוד של הפרויקט הכולל ניתן להוריד מGitHub בקישור הזה.

3 רקע

3.1 ארכיטקטורת Data Lake

המונח Data Lake הוטבע לראשונה על ידי ג'יימס דיקסון, מנכ"ל של חברה בשם Pentaho. משמעות המונח היא מערכת או מאגר של נתונים שנשמרים בצורתם הגולמית/טבעית (לרוב קבצים או מידע בינארי).

סוגי המידע הכלולים בתוך data lake הינם רבים החל מטבלאות ממאגרים רלציוניים עד טקסטים חופשיים ומידע בינארי. כל אלה נשמרים ומועברים לתהליכי המשך כגון: ויזואליזציה, מדידה, אנליטיקה מתקדמת, ולמידת מכונה. כיום קיימים פתרונות On-Premise וגם פתרונות ענן עבור הקמת data lakes כאשר פתרונות הענן העיקריים ניתנים על ידי אמזון, מיקרוסופט וגוגל.

3.2 מגפת הקורונה

מגפת הקורונה היא מגפה עולמית מתמשכת של נגיף COVID-19. הנגיף הינו מזן SARS-CoV-2 וכיום הדביק מעל 110 מיליון אנשים שנמצאו חיוביים, כאשר מתוכם ישנם מעל 2 מיליון מתים. המגפה פרצה לראשונה בדצמבר 2019 בעיר ווהאן שבסין, והחל מאמצע פברואר 2020 החלטה להתפשט במהירות בכל רחבי העולם תוך יצירת בהלה בקרב אזרחי העולם ומשבר כלכלי משמעותי.

על מנת להתמודד עם קצב ההדבקה של הנגיף והסיכון שהוא מביא איתו נערכו ניסיונות מקבילים לפיתוח של מעל 200 סוגי חיסון שונים. בטווח הזמנים בין דצמבר 2020 לינואר 2021 אושרו מספר חיסונים לשימוש על ידי רשויות הבריאות העולמיות והחל מבצע חיסונים עולמי ובפרט בישראל. יעילות החיסונים מול הדבקה בנגיף ואל מול ההשפעה שלו על המחוסנים עדיין נבדקת ונחקרת בימים אלה.

3.3 אינדקסים הסתברותיים

אינדקס הוא מבנה נתונים אשר בא לייצג את תוכנו של מאגר בצורה מהירה לחיפוש ויעילה באחסון ככל האפשר על מנת לא להגדיל את כמות הזיכרון/אחסון הנדרשת עבור אותו מאגר. אינדקס זה מיועד לביצוע שליפות באופן מהיר אשר יביא לתוצאות מדויקות שיספקו את צרכי הלקוח/משתמש.

במצבים מסוימים, התוצאות המוחזרות על ידי אותן שליפות לא מוכרחות להיות מדויקות אלא להיות "קרובות". לדוגמה נתבונן בדוגמה הבאה: כאשר לקוח עם מאגר מידע גדול רוצה לדעת את כמות הרשומות במאגר, ברוב המקרים יספיק עבורו לדעת את גודל המאגר עם

טווח טעות של אלף רשומות מאחר והמאגר כולו מכיל מיליוני ואפילו מיליארדי רשומות. דוגמה זו, ועוד רבות כמוה יכולות לבוא לידי ביטוי על ידי אינדקסים אשר מייצגים תוצאה מקורבת על ידי שימוש מינימלי באחסון. בפרויקט זה בוצע שימוש באינדקסים מסוג זה על מנת לייעל את תהליכי החיפוש והאגירה ללא פגיעה משמעותית בביצועים או גודל האחסון.

4 על הפרויקט

4.1 מוטיבציה

4.1.1 אינדקס חסכוני ויעיל על תוכן של Data Lake

לאור חומרת המגפה והתפשטותה גופים רבים מייצרים נתונים המתארים את המגפה והשפעותיה. נרצה לקחת את כלל הנתונים הקיימים לרשותנו, לקבץ אותם למקום אחד המאפשר פעולות המשך וניתוחים מתקדמים על מנת לייצר תמונת מצב טובה יותר על המגפה. מקום זה הינו data lake המורכב מכמות גדולה מאוד של נתונים.

על מנת שה-Data Lake יאפשר שימוש בתוכנו באופן אפקטיבי ויישמר כך לאורך זמן יש צורך ביצירת אינדקס על התוכן אשר מאפשר ביצוע חיפושים מהירים על גבי המידע תוך שמירה על חסכון במקום כך שהאינדקס לא יגרום לניפוח משמעותי בנפח הData Lake ויוביל לעלויות גבוהות של שימוש באחסון.

4.1.2 אינדקס חסכוני ויעיל על Metadata של קבצים

נוסף על הרצון להיות מסוגלים לבצע ניתוחים מעל תוכן הנתונים נרצה גם לאפשר שקיפות וניתוח של ה-metadata[1] של הנתונים שייכנסו ל-data lake שלנו. הmetadata הוא כלי מאוד חשוב להבנת קטגוריות התוכן הנכנסות למאגר כמו גם למקורות המידע הנכנסים אליו. אינדקס על metadata מוכיח ערך גדול עבור שימושים עם קבצים מולטימדיה (סרטונים, תמונות, מוזיקה ועוד) מאחר וקבצים מסוג זה מכילים המון מידע על תוכנם ועל מקורם בתוך הmetadata שלהם.

בשונה מאינדקס על תוכן, בmetadata לא נרצה הצבעה לקבצים המכילים את פריטי המידע אלא נהיה מעוניינים לענות של שאלות סטטיסטיות על התוכן, לדוגמה:

- כמה פעמים קיים סוג metadata מסוים בתוך המאגר?
- כמה ערכים שונים יש במאגר עבור קטגוריית metadata מסוימת במאגר?

במקרים אלו מספיקה תשובה מקורבת לאור כמות הנתונים ולכן נוכל להשתמש באינדקסים הסתברותיים על מנת לענות על השאלות האלו באופן מדויק מספיק אך חסכוני מאוד בזיכרון ואחסון.

[1] מדובר על ניתוח של מבנה הקבצים, שדות שמגיעים כחלק ממבנה הקובץ, שמות עמודות בטבלאות וכיוצא בזאת.

4.2 תכולות ופיצ'רים

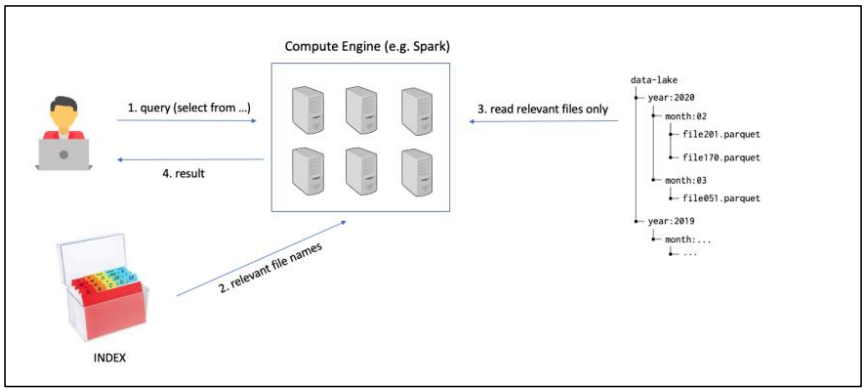
4.2.1 אינדקס תוכן

אחד השימושים הנפוצים עבור data lake הינו חיפוש על תוכן הקבצים ומציאת פרטי המידע הרלוונטיים מתוך כמות המידע הגדולה הקיימת במאגר. ב-data lakes כיום נדרש מעבר על כלל הקבצים במאגר על מנת למצוא מתוכם את הקבצים הרלוונטיים וגם נדרש בתוך כל קובץ רלוונטי למצוא את החלק המתאים לאינדקס בתוך הקובץ.

על מנת לייעל ולהאיץ את תהליך החיפוש המתואר מעלה המאגר מכיל מימוש של האינדקס ממאמר [10] המאפשר חיפוש ערכים על גבי בקבצים טבלאיים ומציאת הקבצים הרלוונטיים ללא צורך במעבר על כל תוכן הקבצים בזמן החיפוש, אלא רק פעם אחת בזמן כניסת הקובץ למאגר.

האינדקס אשר בא לידי ביטוי במאגר שם דגש על חיסכון באחסון על מנת לא להגדיל את ה-data lake באופן משמעותי מעבר לקבצים הקיימים בו ועם זאת מאפשר שליפה מהירה מאוד ביחס לכמות הקבצים ומציאת התוצאות. האינדקס עצמו מאוחסן גם הוא כחלק מתשתית האחסון של ה-data lake ומיוצג על ידי קבצי טקסט אשר משמשים לטובת ביצוע השליפות המתוארות.

למעשה, הארכיטקטורה של האינדקס תיראה כך:



איור 1: ארכיטקטורת High Level של data lake למטרות שאילתות זולות ומהירות

באיור 1 ניתן לראות כי מנוע ה-compute שישרת את בקשות המשתמש לתוכן יבצע שימוש באינדקס המתואר ובעזרתו יוכל לחפש את התוכן אך ורק בקבצים הרלוונטיים מתוך המאגר כולו ובכך יפחית את זמן החישוב וההרצה של שאילתה מהמשתמש.

השליפות המאופשרות על ידי אינדקס זה הן שאילתות SELECT (כמו בשפות SQL) פשוטות מהצורה הבאה:

```
SELECT C1, C2, ...
FROM DATA-LAKE
WHERE SOME-COLUMN = SOME-VALUE
```

על מנת לבצע זאת נסמן את ה-data lake באות D ונייצג אותו כקבוצה של זוגות באופן הבא:

$$D = \{< c1 : v1, c2: v2, ..., cn : vn >| \forall 1 \leq i \leq n, ci \in C, vi \in V \}$$

כאשר C היא קבוצת כל שמות העמודות ו-V היא קבוצת כל הערכים בעמודות. במקום לסרוק את כלל הקבצים באופן מקבילי כפי שנהוג היום במנועי תשאול מעל data lakes אנחנו נרצה למצוא אך ורק את הקבצים הרלוונטיים בעזרת אינדקס ריכוזי ולגשת רק אליהם ובכך להפחית את זמן העיבוד של החיפוש על גבי הקבצים.

בארכיטקטורה שהוצעה במאמר הוצע שיפור שניתן לממש בעזרת אלגוריתם Bloom Filter [1] אותו נסקור גם כן במסגרת עבודה זו. האלגוריתם מאפשר לבדוק האם איבר מסוים שייך לקבוצה עם טעות חד צדדית. אם האלגוריתם מחזיר שהאיבר לא קיים בקבוצה הוא בוודאות לא קיים, אך אם מחזיר כי קיים אז ישנו סיכוי שהאיבר קיים וסיכוי אחר שהאיבר אינו קיים. לאור יעילותו באחסון של האלגוריתם ודיוקו היחסי הוא מאפשר חיסכון נוסף במשאבים ובכך לייעל עוד יותר את הארכיטקטורה המוצעת במאמר [10].

נוסף על כך, בוצעה הרחבה של caching אשר גם היא מוצעת במאמר. הרחבה זו באה לאפשר שיפור ביצועים משמעותי כאשר ניגשים למידע עם ערכים דומים בכמה וכמה קבצים על ידי בדיקת המיפוי עבור ערך מסוים בעלות מהירה וזולה יותר במאגר מצומצם של ערכים השמורים בזיכרון. על מנת לבצע זאת בחרנו בפתרון caching אשר מאפשר גם scale-out של תהליכי הETL שלנו על ידי יצירת cache בטכנולוגיית Redis אשר מאפשר שיתוף cache בין instances של שירות האינדקס של הפרויקט.

בפרויקט זה בוצע מימוש אינדקס תוכן עבור קבצי CSV ו-JSON בלבד.

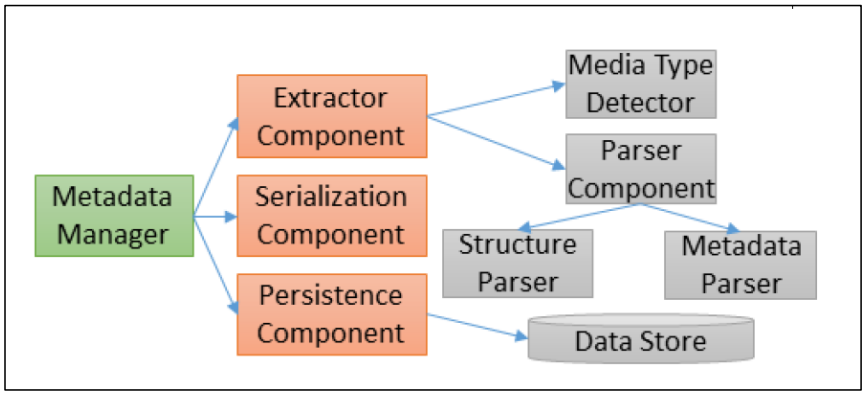
4.2.2 אינדקס Metadata

נוסף על כך נרצה לאפשר ניתוחים מעל ה-metadata[2] של הנתונים שייכנסו ל-data lake שלנו על בסיס הארכיטקטורה שמוצעת ל-GEMMS [5]. על מנת למנוע מה-data lake שלנו להפוך ל-data swamp[3] נרצה לשמור את ה-metadata של הקבצים בתוכו במטרה לשלוט בתוכן ובאיכותו. ארכיטקטורת GEMMS מיועדת לאפשר לחלץ את ה-metadata על שלל

[2] מדובר על ניתוח של מבנה הקבצים, שדות שמגיעים כחלק ממבנה הקובץ, שמות עמודות בטבלאות וכיוצא בזאת.

[3] Data lake אשר לא מנוהל טוב ולא נשלט וכתוצאה מכך חווה ירידה בביצועים ובאיכות המידע בתוכו

סוגי קבצים, לנתח את הנתונים המחולצים ולשמור אותם בצורה נוחה לתשאול ולתיעוד.
הארכיטקטורה המוצעת מוצגת באיור 2.



איור 2: ארכיטקטורת High Level של תהליך חילוץ, ניתוח ושמירה ה-metadata של הקבצים ב-data lake

החיצים מתארים יחס של "משתמש ב" והאיור כולו מציג את תהליך חילוץ ושמירת ה-metadata. תחילה נחלץ את הנתונים מהקבצים על ידי זיהוי סוג המדיה, ניתוח הנתונים ועיבודם, לאחר מכן הצרנה לשמירה במאגר ושמירתם במאגר עם persistent storage.

בנוסף לשמירת ה-metadata עצמו נבצע שני סוגים של ניתוחים הסתברותיים על גבי הנתונים:

- שימוש באלגוריתם HyperLogLog [2] לטובת חישוב מוערך של כמות הפריטים הייחודיים בכל שדה ב-metadata תוך שימוש בזיכרון נמוך מאוד.
- שימוש באלגוריתם Count-Min Sketch [3] לטובת יצירה של טבלת שכיחות מוערכת של הערכים בשדות של ה-metadata תוך אפשרות להערכה יתרה של הכמות, אך לעולם ללא הערכה בחסר.

בפרויקט זה בוצע מימוש אינדקס metadata עבור כל סוגי הקבצים.

4.2.3 שאילתות בפרוטוקול REST

בעזרת האינדקסים המתוארים מעל נבנה שני סוגים של שאילתות מעל ה-data lake שלנו:

- שאילתות על התוכן בפורמט SELECT פשוט כפי שהוצג מעלה
- שאילתות סיכומיות על גבי ה-metadata של הקבצים שלנו
  - באופן רגיל מבוסס שאילתת SELECT
  - באופן סיכומי מבוסס הניתוחים ההסתברותיים שימומשו

על מנת לחפש בתוכן של המאגר נוכל לבצע שאילתה באופן הבא:

1. נבצע בקשת POST לשרת לנתיב /api/v1/Query?queryType=NeedleInHaystack
2. בתוכן הבקשה נשלח אובייקט הבנוי באופן הבא:
    a. Conditions – מערך של אובייקטים המייצגים תנאים באופן הבא:
        i. ColumnName – שם העמודה בה נחפש ערך מסוים
        ii. Value – הערך אותו נחפש בעמודה
    b. Relation – סוג היחס בין התנאים. האופציות הקיימות כיום הן And ו-Or
3. נקבל תוצאה המכילה את כל הקבצים העונים על סט התנאים בבקשה

דוגמה לאובייקט הבקשה:

```
{
  "Relation": "And",
  "Conditions": [
    {
      "ColumnName": "is_first_Test",
      "Value": "Yes"
    }
  ]
}
```

דוגמה לאובייקט התשובה:

```
[
  {
    "FileName": "__CONTENT__/corona_output.csv",
    "HitValues": [
      "Yes"
    ]
  }
]
```

בנוסף, קיימים היום שני סוגים של ניתוחים על הmetadata שניתן לבצע על המאגר:

- Cardinality – הערכה של כמות הערכים הייחודיים בקטגוריית metadata מסוימת
- Frequency – הערכה של כמות החזרות של ערך מסוים בקטגוריית metadata מסוימת

עבור ניתוח Cardinality נוכל לבצע שאילתה באופן הבא:

1. נבצע בקשת POST לשרת לנתיב /api/v1/Query?queryType=Cardinality
2. בתוכן הבקשה נשלח אובייקט הבנוי באופן הבא:
    a. MetadataKey – קטגוריית הmetadata אותה נרצה לנתח
3. נקבל תשובה המכילה את הכמות המוערכת של ערכים ייחודיים בקטגוריה זו

דוגמה לאובייקט הבקשה:

```
{
  "MetadataKey": "csv:delimiter"
}
```

דוגמה לאובייקט התשובה:

```
[
  {
    "Cardinality": 1
  }
]
```

עבור ניתוח Frequency נוכל לבצע שאילתה באופן הבא:

1. נבצע בקשת POST לשרת לנתיב /api/v1/Query?queryType=Frequency
2. בתוכן הבקשה נשלח אובייקט הבנוי באופן הבא:
   a. MetadataKey – קטגוריית הmetadata אותה נרצה לנתח
   b. MetadataValue – הערך עבורו נרצה לבדוק את כמות החזרות
3. נקבל מערך תשובה המכילה את הכמות המוערכת של החזרות של ערך זה בקטגוריה שנבחרה

דוגמה לאובייקט הבקשה:

```
{
  "MetadataKey": "csv:delimiter",
  "MetadataValue": "comma"
}
```

דוגמה לאובייקט התשובה:

```
[
  {
    "Frequency": 1
  }
]
```

4.3 סביבת פיתוח ומימוש הפרויקט

4.3.1 טכנולוגיית Microsoft .NET

.NET היא פלטפורמה חופשית מבית חברת מיקרוסופט, מבוססת קוד פתוח וחוצה-פלטפורמות, שבאמצעותה ניתן לכתוב ולהריץ תוכנות. מיקרוסופט מספקת בעזרת פלטפורמה זו מספר כלים לתוכנות שרצות מעליה, כגון: ניהול זיכרון, שימוש בהתקני קלט/פלט, הצפנת הודעות ועוד. באופן זה מתכנתים המשתמשים ב- .NET אינם נדרשים לכתוב ספריות עזר או קוד ייעודי עבור היכולות והשירותים האלו, ומתאפשר להם להתמקד אך ורק בכתיבה הלוגיקה המרכזית של התוכנה.

כיום פלטפורמה זו מאפשרת להריץ את אותו הקוד במספר מערכות הפעלהשונות: Microsoft Windows, Linux, MacOS, מכשירי טאבלט, טלפונים סלולריים ועוד. יכולת זאת מתאפשרת מכך שהקוד אשר המפתחים כותבים מתקשר אך ורק עם סביבה וירטואלית שמתקבלת מהפלטפורמה, ועבור כל מערכת הפעלה מפותחת סביבה וירטואלית מתאימה ובכך הקוד מנותק מהמימוש של מערכת ההפעלה.

כחלק מפלטפורמת NET. כלולה בין השאר ספריית ASP.NET Core אשר מאפשרת הרצת שרתי web בקלות ובמהירות תוך תמיכה בביצועים גבוהים ובפרוטוקול REST. בעזרת ספריה זו הפרויקט מריץ שירות שאילתות על גבי המאגר. בנוסף, כלולות בפלטפורמה גם יכולות כתיבת לוגים, ניהול תצורה ועוד המקלות את כתיבת הפרויקט ובוצע בהן שימוש רב על מנת להקל על כתיבת הפרויקט.

4.3.2 שירותי Amazon Web Services (AWS)

AWS היא יחידה עסקית של חברת Amazon העוסקת באספקת שירותי מחשוב ענן ליחידים, לחברות ולגופים ממשלתיים. הטכנולוגיה של מוצריה של AWS מאפשרת שימוש בשירותי מחשוב ענן כגון יצירת מכונות וירטואליות, אחסון, מאגרי מידע ועוד המון בצורה פשוטה אשר לא דורשת מהמפתחים ומהחברות לנהל עבור עצמן את כל המשאבים האלו ובכך על ידי תשלום לחברת Amazon מקבלים שירות מנוהל ומנוטר המספק את היכולות לפי דרישה.

בפרויקט זה בוצע שימוש בשירות האחסון S3 של אמזון המאפשר אחסון מידע בצורה מנוהלת ויעילה בענן. שירות האחסון הזה הוא הבסיס לData Lake בליבת הפרויקט ובעזרתו מומשה הלוגיקה עבור אחסון המידע והאינדקסים שלו.

4.3.3 תשתית Apache Kafka

Apache Kafka היא פלטפורמת תוכנה לעיבוד זרם נתונים (stream processing) המבוססת קוד פתוח בשפות Scala ו-Java ומנוהלת במסגרת קרן התוכנה של Apache. הפרויקט נועד לספק יכולת תקשורת בין ספקי מידע (Producers) המייצרים מידע בכמות גבוהה לבין צרכני מידע (Consumers) הצורכים מידע זה לטובת עיבוד ופעולות המשך. תקשורת זו מבוצעת בצורה אסינכרונית ובכך מאפשרת ניתוק תלות בין שני הרכיבים ומאפשר עיבוד נתונים בכמויות גדולות ובקצבים מהירים מאוד. Kafka מבוססת על ארכיטקטורה מבוזרת הניתנת להגדלה (scalable).

בפרויקט זה בוצע שימוש בKafka על מנת לתקשר בין שלב העלאת הקבצים ל-Data Lake לבין פעולות עדכון האינדקסים בצורה אסינכרונית המאפשרת עיבוד באצווה (Batch Processing) ובכך לייעול תהליכי הכנסת המידע ל-Data Lake.

4.3.4 מאגר Redis

Redis הוא בסיס נתונים מסוג NoSQL הפועל בתוך הזיכרון (In-Memory) ומבוסס קוד פתוח. השימושים בטכנולוגיה של Redis הינם מגוונים וכוללים אחסון מידע בתצורת Key-Value באופן מבוזר, Caching, והעברת הודעות בין שירותים.

בפרויקט זה בוצע שימוש בטכנולוגיה של Redis לטובת cache מבוזר לתוכן האינדקס המאפשר ייעול ביצועים עבור שירות הרץ בכמה instances במקביל ומאפשר גדילה של השירות (scalability) בצורה רחבה.

4.3.5 ספריית Apache Tika

Apache Tika היא פלטפורמה לזיהוי תוכנה וניתוחו המבוססת קוד פתוח בשפת Java ומנוהלת במסגרת קרן התוכנה של Apache. הפלטפורמה מאפשרת ניתוח תוכן של קבצים, חילוץ metadata ממגוון מאוד רחב של סוגי קבצים ועוד.

בפרויקט זה בוצע שימוש בפלטפורמה לטובת חילוץ metadata עבור כל הקבצים הנכנסים ל-Data Lake על מנת להוסיף מידע זה לאינדקס ולבצע ניתוחים לאחר מכן. ניתן להריץ את Tika בשתי תצורות על מנת לחלץ metadata של קובץ:

[EG1] **עם הערות:** חסרה לי כאן דוגמא כיצד מחלצים את המטהדטה מתוך קובץ. מספיק דוגמא מסוג קובץ אחד

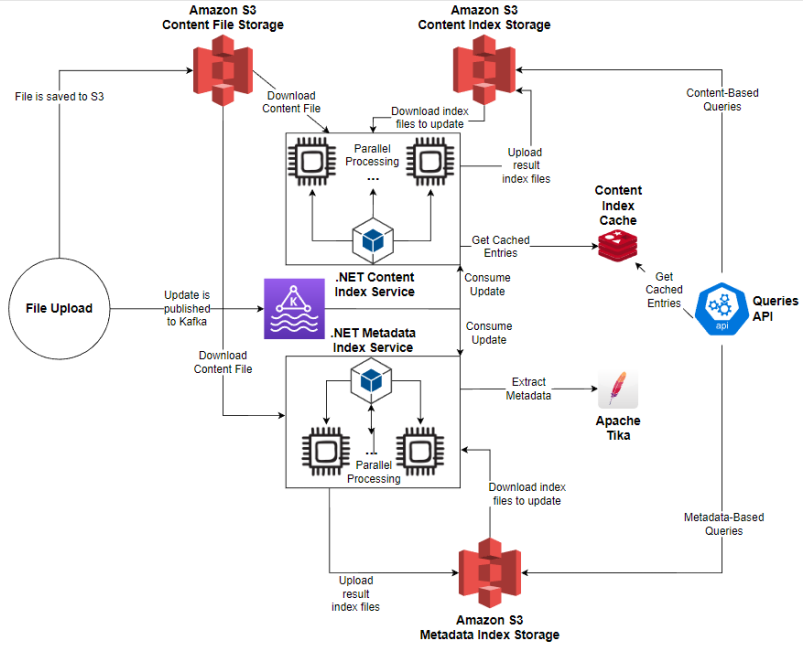
1. כשרת standalone שניתן להעלות אליו קבצים ולקבל את ה-metadata על הקובץ. דוגמה לתשובה שמתקבלת בבקשה כזאת עבור קובץ CSV:

```
Content-Encoding,windows-1252
language,en
csv:delimiter,comma
Content-Length,41783
Content-Type-Override,text/csv
Content-Type,text/csv; charset=windows-1252; delimiter=comma
```

   מה שניתן לראות כאן הוא שדות המתארים את הקובץ, כגון הקידוד שלו, אורך הקובץ, סוג התוכן שלו ועוד.
2. על ידי שימוש בספריית Tika הכתובה בשפת Java ולשלוח את הקובץ הרצוי לניתוח בספריה. תצורה זו הינה יותר יעילה מבחינת ביצועים מאחר ואין צורך להעלות את כל הקובץ לשרת וניתן לגשת אליו באופן מקומי בדיסק. התשובה המתקבלת לאחר ניתוח הקובץ בעזרת הספריה מכילה את אותו תוכן כפי שראינו בתצורת השרת אך המידע מובנה לתוך אובייקט המוחזר מה-API של הספריה.

4.4 ארכיטקטורת המערכת

4.4.1 ארכיטקטורת על



איור 3: ארכיטקטורת המערכת

כפי שניתן לראות באיור מעלה יש שימוש במספר רכיבים וטכנולוגיות לטובת מימוש דרישות הפרויקט. תחילה נתאר את תהליך זרימת המידע במערכת לפי השלבים באיור:

1. מעלים קובץ ל-Data Lake. ההעלאה יכולה להתבצע באופן ידני ל-S3 או בעזרת ה-API שנוצר עבור זה.
2. לאחר השלמת ההעלאה נכתב עדכון על הוספת הקובץ ל-topic של Kafka המציין את שם הקובץ המלא בתוך הS3.
3. כעת קורים שני תהליכים באופן מקביל ובלתי תלוי:
   a. עדכון אינדקס התוכן של ה-Data Lake בהתאם לקבצים החדשים שנוספו. שירות אינדקס התוכן יודע לעבד כיום קבצי json ו-csv באופן מקבילי על מנת לעמוד בביצועים הנדרשים. עדכון זה כולל הורדה של קבצי התוכן וקבצי אינדקס קיימים לטובת העדכון ושימוש ב-cache מבוסס טכנולוגיית Redis אשר גם משמש כמנגנון סנכרון ונעילות בין תהליכים. לאחר השלמת עדכון האינדקס באופן מקומי, קבצי האינדקס המעודכנים עולים ל-S3 הייעודי עבור קבצי האינדקס של תוכן ה-Data Lake.

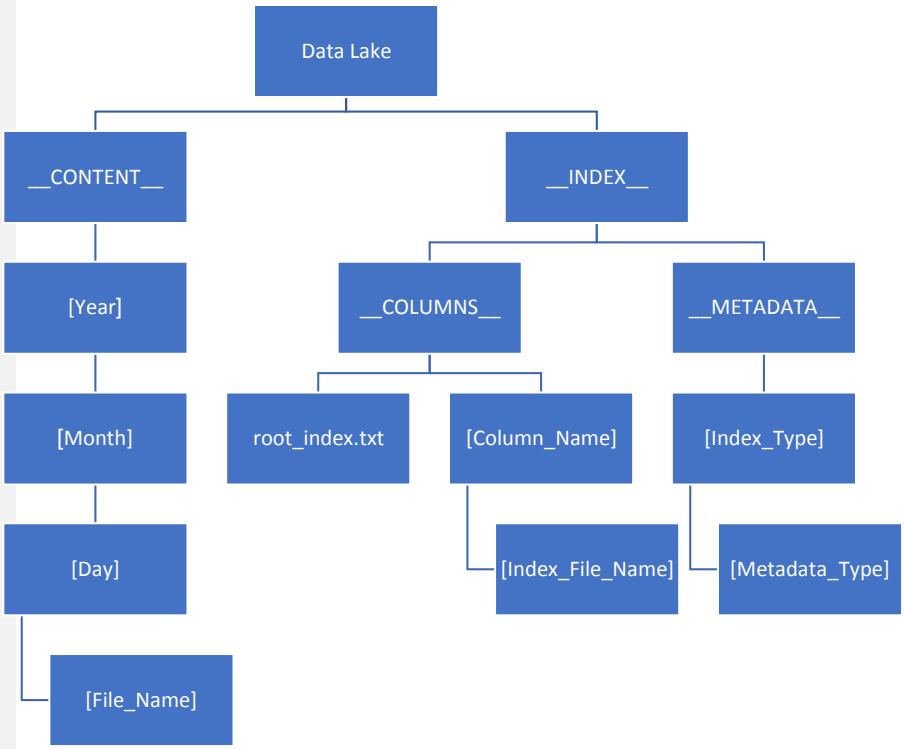
b. עדכון אינדקס ה-metadata של ה-Data Lake בהתאם לקובץ החדש שנוסף. השירות נעזר בפלטפורמה של Apache Tika לטובת חילוץ ה-metadata מהקובץ החדש ולאחר מכן מוריד את קבצי האינדקס הקיימים ומעדכן אותם בהתאם לקובץ החדש. תהליך האינדקס של ה-metadata מתבצע באופן מקבילי על גבי מספר ליבות על מנת לאפשר ביצועים טובים.

4. לאחר השלמת עדכון האינדקסים ניתן לתשאל על הקובץ החדש באמצעות ה-Queries API שנכתב במסגרת הפרויקט ומאפשר תשאול המאגר על בסיס האינדקסים שעודכנו על ידי מספר סוגי שאילתות שהוצגו מעלה.

4.4.2 מבנה מאגר ה-Data Lake

על מנת לבנות את מאגר ה-Data Lake נדרש תכנון מפורט של מבנה הקבצים עבור התוכן ועבור האינדקסים על מנת לאפשר לנהל את הקבצים בצורה נוחה ולאפשר גדילה של המאגר באופן טבעי ללא צורך בשינוי קוד והתנהגות.

מבנה המאגר מתואר על ידי איור 4 המצורף מטה:



איור 4: מבנה המאגר

ניתן לראות באיור 4 כי יש הפרדה בין האינדקסים על התוכן לבין התוכן עצמו על ידי שימוש בתיקיות שונות בתוך המאגר. הפרדה זו מיועדת כדי לאפשר ניתוק תלות טכנולוגית בין התוכן לבין האינדקסים במקרה ונרצה לשנות את המימוש הטכנולוגי של אופן אחסונם.

מבנה תיקיית התוכן מבוסס על תאריך הכנסת הקבצים למאגר. מבנה זה מאפשר לשלוט בקלות על זמן האגירה של התוכן, ובמקרה הצורך נוכל להחליט למחוק טווח זמנים מסוים שנמצא בעבר ולהוריד בעלויות האחסון. בנוסף, חלוקה זו מאפשרת לעקוב אחרי העלאת קבצים בקלות מאחר ונוכל לחפש אותם בתוך התאריך של יום ההעלאה ובקלות להבין אם הגיעו או לא.

גם בתוך התיקיה של האינדקסים אנו רואים הפרדה בין אינדקסים על התוכן של הקבצים לבין אינדקסים על metadata של הקבצים. הפרדה זו באה מאותו עיקרון של הפרדה בין התוכן לבין האינדקסים, על מנת לאפשר הפרדה טכנולוגית בין סוגי האינדקסים ואפשרות עתידית של אחסון בדרכים שונות של אינדקסים אלו.

אינדקס התוכן נמצא בתוך תיקיית __COLUMNS__ ומחולק לפי עמודות בקבצי התוכן אשר עולים למאגר. כל תיקייה עבור עמודה מכילה בתוכה את כל האינדקס עבור עמודה מסוימת, כאשר האינדקס יכול להיות קובץ אחד או מספר קבצים, תלוי בכמות המידע המוכל באינדקס. מעל כל התיקיות של העמודות ישנו קובץ root_index.txt המכיל את המיפוי בין טווחי הערכים והעמודות לבין הקובץ המדויק בו יהיה את המידע הרלוונטי. מבנה זה תואם למבנה המוצע במאמר [1].

אינדקס ה-metadata נמצא בתוך תיקיית __METADATA__ ומחולק לפי סוגי האינדקס הקיימים עבור metadata. כיום קיימים שני סוגי אינדקסים ל-metadata:

1. CMS_FREQUENCY – שימוש במבנה של Count-Min-Sketch [2] על מנת להעריך את התדירות של ערך מסוים בתוך שדה metadata נבחר.
2. HLL_CARDINALITY – שימוש במבנה של HyperLogLog [3] על מנת להעריך את כמות הערכים הייחודיים עבור שדה metadata נבחר.

בתוך כל תיקייה עבור סוג אינדקס כזה ישנו קובץ עבור כל סוג שדה metadata הקיים בקבצים אותם נעלה למאגר שלנו.

4.5 דרכים להרחבה

4.5.1 שאילתות תוכן מורכבות

כיום הפרויקט מאפשר ביצוע שאילתות תוכן עם תנאים פשוטים של "וגם" ו"או" עם היררכיה אחת בלבד על תנאים של אופרטור שוויון בלבד. לאור כך אני מאמין שניתן לבצע הרחבות באופן הבא:

1. היררכיה מורכבת לשאילתה אשר תאפשר חיבור מספר פרמטרים לכדי תוצאות מדויקות יותר על גבי המאגר
2. אפשרות לתשאול על טווחים – למשל "מצא לי את כל הקבצים בהם יש בעמודה X ערך כל שהוא בין Y ל-Z". יכולת זאת דורשת מימוש מורכב על גבי האינדקסים הקיימים אשר יכול להיות מעניין לבחינה עתידית.

4.5.2 ביצועים

חלק משמעותי מפרויקט זה היה עבודה על ביצועים טובים ומהירים עבור השירותים והמנועים הכלולים בפרויקט. למרות העובדה כי כיום ביצועי המנועים הינם מהירים מאוד אל מול כמויות התוכן אני מאמין שניתן לבצע שיפורים נוספים בעזרת caches נכונים ואופטימיזציות נוספות, כגון:

1. הוספת מנגנון cache לקבצי האינדקס של התוכן שיאפשר לשאילתות שחוזרות על עצמן להחזיר תשובה בזמן מהיר יותר
2. בחינת מנגנוני parallelization נוספים שיאפשרו ריצה מהירה יותר על קבצים
3. ריצה בסביבת ענן בתשלום אשר נמצאת באותו region של AWS בענן של חברת Amazon על מנת לשפר את ביצועי הרשת של הורדת הקבצים ועדכונם. ריצה בסביבה כזו תדרוש כוח מחשוב גבוה ולכן לא נכללה במסגרת הפרויקט שרץ על פרויקט חינמי של AWS.

4.5.3 פורמטים נתמכים נוספים

קבצי האינדקס המיועדים לתוכן הינם רכיב מרכזי וליבתי בפרויקט זה, אך עם זאת הפורמטים היחידים לקבצי תוכן אשר ניתן להכניס לאינדקס זה הם קבצי CSV ו-JSON. לאור הארכיטקטורה בה נבנה הפרויקט ניתן להוסיף תמיכה בפורמטים נוספים על בסיס ספריה חיצונית או לוגיקה פנימית כל עוד היא קיימת.

4.6 מאגרי מידע לטובת Datasets

4.6.1 מאגר משרד הבריאות הישראלי

קישור ל-dataset: https://data.gov.il/dataset/covid-19

גודל ה-dataset: 2.2 GB

משרד הבריאות אחראי לבריאות הציבור ולמערכות שתפקידן למנוע ולאבחן מחלות, לטפל בחולים, ולקדם את הבריאות פעילות המשרד מתמקדת בחקיקה, רישוי שירותים ובעלי מקצוע, קביעת תקנים לתרופות, מזון ותכשירים, ופרסום מידע לציבור המשרד מספק שירותי בריאות. במסגרת המאמצים למלחמה במגפת הקורונה במדינת ישראל תועדו הנתונים ב3 פורמטים של קבצים: CSV, PDF, XLSX.

קבצים אלו הועלו לאתר מאגרי המידע הממשלתיים (https://data.gov.il) לטובת צריכה של הקהל הרחב ולשימושו. הקבצים כוללים מידע על נתוני בדיקות מעבדה, בידודים, מאפיינים של הנדבקים, החלמות מקורונה ועוד. כל אלו מספקים תמונת מצב רחבה על התפרצות מגפת הקורונה במדינת ישראל ועל ההתפתחות לאורך הזמן. לאור דעיכת המגיפה במדינה כיום המאגר בתחזוקה נמוכה ורוב הקבצים כבר אינם מתעדכנים עם נתונים עדכניים.

4.6.2 מאגר המרכז האירופי לבקרת מחלות ומניעתן

קישור ל-dataset: https://www.ecdc.europa.eu/en/covid-19/data

גודל ה-dataset: 0.1 GB

המרכז האירופי לבקרת מחלות ומניעתן היא סוכנות עצמאית של האיחוד האירופי שתפקידה לחזק את ההגנה של אירופה מפני מחלות זיהומיות. המרכז הוקם בשנת 2004 ונמצא בעיר סולנה שבשוודיה.

החל ממרץ 2021 הסוכנות מעלה עדכונים שבועיים של הנתונים על התפרצות הקורונה ביבשת אירופה ומאפשרת גישה אליהם. הקבצים זמינים בפורמטים הבאים: CSV, XLSX, JSON, XML.

4.6.3 מאגר CORD-19

קישור ל-dataset: https://www.kaggle.com/datasets/allen-institute-for-ai/CORD-19-research-challenge

גודל ה-dataset: 0.6 GB

פרויקט Semantic Scholar המופעל על ידי Allen Institute for AI הוא כלי חיפוש המבוסס על AI למטרות מחקר וחיפוש של כתבי עת ומאמרים מדעיים. הפרויקט יצא לאור בנובמבר 2015 ומשתמש ביכולות NLP מתקדמות על מנת לספק סיכומים של מאמרים אקדמיים. נכון לספטמבר 2022 המאגר של Semantic Scholar כולל מעל 200 מיליון פרסומים מכל עולמות המחקר המדעיים.

בתגובה למגפת הקורונה, הבית הלבן בשיתוף עם קבוצות מחקר מובילות עמלו והכינו את ה- COVID-19 Open Research Dataset (CORD-19). כיום CORD-19 מכיל מעל מיליון מאמרים אקדמיים, כולל מעל 400 אלף עם טקסט מלא, כאשר כולם עוסקים בנושאים הקשורים למגפת הקורונה. המאגר הזה מסופק בחינם לטובת קהילת המחקר הגלובלית על מנת להשתמש ביכולות NLP מתקדמות עם מידע זה ובכך לקבל תובנות חדשות אל תוך המידע אשר יוכלו לסייע במאבק נגד מגפה מדבקת זו.

המאגר כולל קבצים בפורמטים הבאים: CSV, JSON, PDF.

4.6.4 מאגר The COVID Tracking Project

קישור ל-dataset: https://covidtracking.com/data/download

גודל ה-dataset: 0.01 GB

The Covid Tracking Project הוא פרויקט לטובת מעקב אחר מגפת הקורונה בארצות הברית. הפרויקט כולל איסוף, וידוא ופרסום של מידע בנוגע למגפת הקורונה מ56 מדינות וטריטוריות בארצות הברית. המידע כולל את האזורים העיקריים הבאים:

- בדיקות קורונה

- אשפוזים בבתי חולים
- תוצאות המחלה (האם נשארו בחיים לאחר המחלה)
- מידע אתני ודמוגרפי בנוגע למחלה
- נתונים על הטיפול ב-Long COVID

ה-dataset מכיל קבצים לכל מדינה בארצות הברית בפורמט CSV, אך עם זאת הפסיק להתעדכן במרץ 2021.

4.6.5 מאגר Google COVID-19 Open Data

קישור ל-dataset: https://health.google.com/covid-19/open-data/raw-data

גודל ה-dataset: 6.7 GB

מאגר המידע הפתוח בנושא מגפת הקורונה המתוחזק על ידי חברת גוגל מספק איחוד ואיסוף מידע מרחבי הרשת בנוגע למגפת הקורונה לטובת שימוש של משתמשים טכניים. המידע שנמצא במאגר מעודכן על בסיס יומי מתוך מאות מקורות מידע שונים.

4.6.6 מאגר AWS public data lake for COVID-19

קישור ל-dataset: https://aws.amazon.com/covid-19-data-lake/

גודל ה-dataset: 0.4 GB

בתור ספקית שירותי ענן מרכזית, חברת Amazon בחרה ליצור מאגר ריכוזי של מידע עדכני ומתעדכן באופן קבוע של datasets בנוגע למגפת הקורונה. המידע הכלול במאגר הינו לפני עיבוד וניתן לקריאה באופן פומבי כך שהוא מוכן לניתוח על ידי כל מי שמבקש לעשות זאת.

מאגר המידע הזה מאוחסן באמצעות שירותי הענן של AWS ומאפשר גישה מהירה וחינמית אליו לכל בעל או בעלת חשבון בשירותי הענן של Amazon.

4.6.7 מאגר DataHub Novel Coronavirus 2019

קישור ל-dataset: https://datahub.io/core/covid-19

גודל ה-dataset: 1.1 GB

DataHub הוא פרויקט שהתחיל על ידי Datopian ו-Open Knowledge International. במשך עשור הפרויקט כלל פיתוח כלים ואפליקציות לטובת שימוש במידע ומחקרים. כחלק מהפרויקט נבנתה פלטפורמת CKAN אשר עומדת מאחורי אתרי מידע גדולים כגון: data.gov, data.gov.uk.

אחד מה-datasets הנמצאים בפרויקט זה הוא Novel Coronavirus 2019 המכיל 10 קבצים עם מידע רבה בנוגע לתחלואה, הבראה בהפרדה לפי מדינה.

ה-dataset כולל 9 קבצים הניתנים להורדה בפורמט JSON או בפורמט CSV.

5 שיטות אינדקס המומשות בפרויקט

5.1 אינדקס Needle in a Haystack

5.1.1 רקע

במאמר [10] מתואר מבנה אינדקס על תוכן הקבצים אשר מאפשר לייעל שליפות תוכן מעל מאגר data lake עם תוכן רב תוך חיסכון באחסון עבור האינדקס ושמירה על ביצועים טובים. נקרא לשיטה זו מעתה והלאה "Needle in a Haystack".

הצלחת מבנה האינדקס הזה תיבחן על ידי הפרמטרים הבאים:

1. גודל האחסון ביחס לגודל קבצים התוכן
2. זמני ההכנסה וההוספה לאינדקס ככל שהאינדקס גדל
3. זמני התשאול על האינדקס

על מנת לבנות אינדקס יעיל לתוכן המאגר נבחרה ארכיטקטורת data lake המבוססת על המאמר [10] כך שתאפשר ביצוע יעיל, מהיר וזול של שאילתות על המידע במאגר עם הארכיטקטורה המוצגת באיור 1.

במבנה אינדקס זה אנחנו רואים חסכון באחסון על ידי השימוש בקבצים פשוטים במקום מבני נתונים הדורשים אחסון בזיכרון, דבר שעולה הרבה יותר כסף. אמנם הביצועים של אינדקס בתוך זיכרון הינם מוכחים להיות יותר יעילים אך עם זאת, לאור הצורך באינדקס במקרה המתואר עבור שליפות על data lake, אשר מצופה מהן לחזור מהר אבל לא באותה מהירות של שליפה על מאגר נתונים סטנדרטי, ניתן לאפשר האטה מסוימת בשליפה שתאפשר חיסכון משמעותי בעלויות הענן. על מנת לבחון את סדרי הגודל של עלויות הענן הנדרשות לאחסון של אינדקס של מאגר נתונים רגיל אל מול אחסון של storage רגיל של קבצים נסתכל על הטבלה הבאה:

	cost of 1TB/year	pricing info references
Cassandra on EC2	16,164 $	[5, 9]
DynamoDB	2,925 $	[4]
S3	276 $	[6]

טבלה 1: השוואה של עלויות ענן עבור מוצרי אחסון שונים לנתונים

כפי שניתן לראות בטבלה 1, עלויות אחסון קבצים במוצר אחסון ענן כמו S3 קטנה בלפחות פי 10 ובמקרים מסוימים פי 80 מאשר מאגר נתונים סטנדרטי אשר רץ בענן מנוהל כמו AWS.

עם כל זאת, השימוש במאגר מבוסס אחסון אובייקטים כמו S3 מביא איתו גם חסרונות, ביניהם:

1. זמן תגובה ארוך: המספרים משתנים בין כל שימוש לשימוש, אך באופן כללי ניתן להגיד שמוצרי אחסון אובייקטים הינם בעלי זמן תגובה ארוך יותר מאשר מערכות DBMS [11].
2. מגבלה על קצב בקשות: בעוד במערכות DBMS קצב הבקשות אשר ניתן להגיע אליו הוא מאוד גבוה וכמעט בלתי מוגבל, במוצרי אחסון אובייקטים ישנה מגבלה עליונה משמעותית (לרוב בסביבות אלפי בקשות בשנייה [12])
3. האחסון מעוצב ובנוי עבור קבצים גדולים: מוצרי אחסון אובייקטים מיועדים לאחסון לרוב לאחסון קבצים גדולים מאוד (לרוב מספר GB [9]) ולכן אינם בנויים עבור מערכות אשר מחפשות לבצע קריאות אקראיות בקבצים.
4. אין אפשרות לכתיבה רנדומלית: הדרך היחידה לבצע שינויים באובייקטים וקבצים שנמצאים במוצרי אחסון אובייקטים היא להחליף אותם במלואם, דבר אשר יכול להוות בעיה עבור מערכות שדורשות כתיבה רנדומלית.

לכל אלו ישנה התייחסות בדרך בה נבנה המאגר והאינדקסים בו כפי שמפורט בהמשך.

5.1.2 מבנה האינדקס

האינדקס בנוי על ידי הפרדה למחיצות שונות לפי שמות העמודות (כפי שניתן לראות באיור 4) ובכך מאפשר שליטה על גודל האינדקס לכל עמודה וביצוע שיפורים נקודתיים לפי עמודה. כל קובץ בתוך מחיצה של עמודה אחראי לייצוג של טווח ערכים מסוים ובכך החיפוש מחולק באופן נוח יותר ומאפשר ריצה על קבצים קטנים יותר.

במאמר המקורי היו 2 חלקים לקובץ: תוכן האינדקס (data) ו-metadata.

במימוש אשר בוצע בפרויקט קבצי האינדקס מכילים 4 חלקים:

1. תוכן האינדקס – עבור כל ערך בטווח הערכים המיוצג על ידי הקובץ נשמר מיפוי לקבצים בהם הוא מופיע. כל ערך מיוצג על ידי שורה בפורמט JSON באופן הבא:
   {"Files":["__CONTENT__/2022/9/27/024da9ea-5f83-41a7-adfc-da421303e388.json"],"Value":"769"}
   השדה Files מכיל את שמותיהם של כל הקבצים המכילים את הערך בשדה Value.
2. Metadata – החלק מייצג טווחים של ערכים המיוצגים בתוך הקובץ על מנת לאפשר חיפוש מהיר בתוך הקובץ. כל טווח ערכים בתוך הקובץ מיוצג על ידי שורה בפורמט JSON באופן הבא:
   {"Offset":4096,"Max":"805","Min":"7871"}

השדה Offset הוא ההיסט בבתים בתוך הקובץ בוא ניתן להתחיל למצוא רשומות של תוכן האינדקס בין הערכים בשדות Min ו-Max.

3. Bloom Filter – מאפשר בדיקה מהירה האם ערך מסוים נמצא או לא נמצא בתוך הקובץ. חלק זה בוצע לפי ההצעה שבמאמר. חלק זה מיוצג בפורמט בינארי אשר ניתן לפירוש רק על ידי הקוד בפרויקט שמאפשר קריאה שלו. בהמשך יורחב עוד בנושא השימוש במבנה הנתונים של Bloom Filter לטובת הפרויקט.
4. הצבעות לחלקי הקובץ – על מנת לאפשר גישה מהירה לשלושת החלקים הראשונים ישנו חלק קטן בגודל 16 בתים המכיל מספר אחד (8 בתים) המצביע לנקודה בקובץ בה מתחיל חלק 2 ומספר נוסף (8 בתים) המצביע לנקודה בקובץ בה מתחיל חלק 3).

עבור data lakes גדולים עם קבצים מרובים האינדקס המתואר מעלה יגדל בהתאם ולבסוף גם בו יהיו הרבה מאוד קבצים אשר ידרשו מעבר ארוך ואיטי על התוכן שלהם בכדי למצוא את הקבצים הרלוונטיים ובכך נאבד היתרון. על מנת להתמודד עם זה נוצר קובץ גדול אשר מהווה אינדקס של האינדקסים ובא לאפשר מיפוי של קבצי האינדקסים לפי טווח הערכים בהם והעמודה אותה הם מייצגים. קובץ זה ייקרא מעתה "root index".

קובץ ה-root index בנוי כך שעבור כל קובץ אינדקס קיימת שורה בפורמט JSON המייצגת אותו באופן הבא:

{"FileName":"__INDEX__/__COLUMNS__/Confirmed/5cc10ec7-297b-4278-b41a-65539da96162.txt","Max":"99999","Min":"9825","ColumnName":"Confirmed"}

[EG2] עם הערות: תסביר את השדות של ה ROOT-IINDEX

השדה FileName הינו שם הקובץ באחסון S3 המכיל את רשומות האינדקס בין הערכים המיוצגים בשדות Min ו-Max עבור העמודה בשדה ColumnName

עבור כל אחד מהקבצים ניתן להגדיר את כמות הערכים בכל קובץ או מקבץ על מנת לאפשר שליטה והתאמה לגודל ה-data lake והתאמה לביצועים הנדרשים.

5.1.3 תהליך עדכון האינדקס

עבור כל קבוצת ערכים חדשה שנכנסת לעמודה מסוימת במאגר נבצע עדכון של האינדקס על מנת שיוכל לייצג את הערכים האלו. השלבים בתהליך העדכון הם:

1. <u>חילוץ הערכים מהקובץ:</u>
   תחילה על מנת לאפשר עדכון נרצה להבין את הערכים החדשים שנוספו. לשם כך עבור כל סוג קובץ נתמך באינדקס ישנה לוגיקת חילוץ ערכים מתוך הקובץ בתקן המאפשר בקלות להוסיף לאינדקס הקיים. ההתייחסות לקבצים היא כקבצים

טבלאיים וחילוץ הערכים מתבצע בהתאם, הערכים המחולצים מחולקים לעמודות וכל עמודה מעובדת בתהליכי ההמשך בנפרד ובמקביל על מנת לייעל את הביצועים.

2. מיפוי הערכים לקבצי האינדקס:

על מנת לעדכן את האינדקס נדרש להבין לאיזה קובץ צריך להתבצע העדכון. לשם כך ניעזר ב-root index אשר מבצע בדיוק את זה. בעזרת ה-root index נמצא עבור כל ערך את הקובץ אליו הוא שייך ונשמור את המיפוי הזה עבור ביצוע העדכון בשלבים הבאים. במימוש שקיים בפרויקט זה תחילה נחפש ב-cache של ה- root index שקיים עבור ביצועים משופרים ואם הערכים לא קיימים שם המנוע יוריד את קובץ ה-root index ויקרא ממנו. במקרה ולא קיים קובץ מתאים עבור הערכים שמצאנו החיפוש בקובץ ה-root index יחזיר ערך ברירת מחדל שישמש לזיהוי תרחיש בו נדרש ליצור קובץ אינדקס חדש

3. יצירת קובץ האינדקס:

בהנחה ואין קובץ אינדקס קיים מתאים עבור קבוצת ערכים מסוימת ניצור קובץ חדש בפורמט שהגדרנו, תחילה נמיין את הערכים שקראנו מקובץ המקור ולאחר מכן נרשום את כל הערכים והמיפוי שלהם לקבצים. לאחר מכן נרשום את ה-metadata עבור קבוצות הערכים בתוך הקובץ על מנת לאפשר חיפוש מהיר יותר על גבי הקובץ. לבסוף ניצור bloom filter חדש עם הערכים שבקובץ על מנת לאפשר חיפוש מהיר יותר. במקרה שמספר הערכים החדשים גדול ממספר הערכים המותר לקובץ אינדקס אנו ניצור מספר קבצים בהתאם לכמות הערכים.

4. עדכון קובץ האינדקס:

במקרה שקיים קובץ מתאים לערכים אותם קראנו נבצע עדכון על האינדקס. העדכון יתבצע על ידי כתיבה מחדש של הקובץ מאחר ואין לנו אפשרות לגישה אקראית לקובץ לאור סוג הקבצים בהם אנחנו עושים שימוש. את העדכון נבצע על ידי מיון הערכים שקראנו מקובץ המקור וקריאתם יחד עם הערכים הקיימים בקובץ האינדקס הקיים. לפני תחילת הכתיבה לקובץ החדש נקרא את ה-bloom filter מקובץ האינדקס הקיים לטובת עדכונו. בשלב זה נרצה לכתוב את האינדקס החדש שורה אחר שורה. עבור כל שורה כזו נשווה בין הערך המינימאלי שטרם נכתב מקובץ האינדקס הקיים לערך המינימאלי שטרם נכתב מקובץ המקור ונבחר ביניהם את הערך המינימאלי ונכתוב אותו לקובץ ונמשיך לשורה הבאה. במקרה והערכים האלו שווים נבצע איחוד בין הערכים ונוסיף את קובץ המקור החדש לרשימת הקבצים הממופים לערך הקיים באינדקס. עבור כל שורה שתיכתב נשמור בזיכרון את שורת ה-metadata המתאימה וכאשר נסיים לכתוב את כל הערכים לקובץ נכתוב את שורות ה-metadata. במקביל לשמירה של שורות ה-metadata הרלוונטיות עבור כל ערך שנכניס לאינדקס נבצע עדכון ב-bloom filter על מנת שיכיל את כל הערכים

[EG3] עם הערות: צריך כאן דוגמא לאינדקס פשוט ולעדכון שלו כמו שנמצא למשל במאמר של גרישה וכן דוגמא לשימוש בפילטר בלום בסעיף 4 להלן

ויהווה ייצוג אמין של תוכן האינדקס. לאחר כתיבת שורות ה-metadata נכתוב את הייצוג הבינארי של ה-bloom filter לקובץ ונשמור אותו לטובת שימוש בהמשך. חשוב לציין שבמקרה שכמות הערכים לאחר האיחוד עם הערכים מקובץ המקור עולה על כמות הערכים המותרת לקובץ אינדקס נפצל את הקובץ החדש למספר קבצים ונבצע עדכון בהתאם.

5. <u>עדכון ה-root index:</u>

כעת כשהעדכונים לקבצי האינדקס בוצעו נרצה לעדכן גם את ה-root index על מנת שיוכל להצביע באופן עדכני ואמין לקבצי האינדקס לפי טווח הערכים. עבור כל קובץ אינדקס חדש ניצור שורה חדשה ב-root index המייצגת את טווח הערכים השמור בקובץ. עבור קובץ אינדקס קיים שעודכן נבצע עדכון לטווח הערכים בקובץ. במקרה של פיצול קובץ אינדקס קיים תמיד יישאר שימוש בשם האינדקס המקורי בלפחות קובץ אחד על מנת להקל את תהליך העדכון של ה-root index בכך שלא נדרש למחוק שורות ישנות, דבר אשר מהווה מורכבות גדולה יותר.

בנספח 2 ניתן לראות דוגמה למצב לפני העדכון ואחרי העדכון.

סיבוכיות זמן ההכנסה לאינדקס מחולקת באופן הבא:

1. חיפוש הקובץ לעדכון ב-root index: O(n) כאשר n הוא מספר הערכים במאגר בכל העמודות. קובץ ה-root index הוא ייצוג מוקטן של הערכים אך עדיין בסדר גודל לינארי ומאחר ובמקרה הגרוע ביותר נצטרך לעבור על הקובץ בשלמותו אזי שנעבור על n*c ערכים כאשר c הוא מספר קבוע בין 0 ל-1 המייצג את כמות השורות שניתן לשמור בקובץ אינדקס בודד. ניתן לבצע הרחבה לטובת שיפור עתידי על ידי יצירת חלק של metadata בסוף הקובץ של root index בדומה לקובץ האינדקס הרגיל ובכך להקטין את סיבוכיות הזמן להיות O(m*k) כאשר m הוא מספר העמודות ו-k הוא מספר הערכים המקסימלי בעמודה במאגר.
   בעזרת השימוש במנגנון caching עבור ה-root index (כפי שמפורט בהמשך) כאשר רשומת האינדקס מקובץ root index כבר קיימת ב-cache סיבוכיות הזמן של חיפוש רשומה ב-root index לוקחת O(log(k)) כאשר k הוא מספר הערכים המקסימלי בעמודה במאגר. זמן זה נובע מהמימוש שנבחר ל-cache בפרויקט בטכנולוגיית Redis.
2. יצירת קובץ האינדקס: O(d) כאשר d הוא מספר הערכים המקסימלי בקובץ שנוסף למאגר.

3. עדכון קובץ אינדקס: O(d) כאשר d הוא מספר הערכים המקסימלי בקובץ שנוסף למאגר. קובץ אינדקס קיים הוא בעל גודל חסום במספר קבוע ולכן לא משפיע על הסיבוכיות כאן באופן משמעותי.
4. עדכון ה-root index: O(n) כאשר n הוא מספר הערכים בכל העמודות בכל המאגר. הסיבה לכך היא שעל מנת לעדכן את ה-root index יש לעבור על כל השורות של הקובץ ולכתוב אותן מחדש ו/או לעדכן אותן בהתאם, זאת בנוסף לשורות חדשות שהתווספו. כפי שתואר בסעיף של חיפוש על גבי קובץ זה מספר השורות בקובץ הוא O(n) ולכן גם העדכון הוא כזה.
5. לכן לבסוף סיבוכיות הזמן במקרה הגרוע היא O(n) כאשר n הוא מספר הערכים בכל העמודות במאגר, אך בזכות העובדה כי ישנו פקטור משמעותי של דחיסה והקטנה של הייצוג בקובץ ה-root index הביצועים המושגים בעזרתו מאפשרים שימוש טוב ומהיר במאגר.

5.1.4 תהליך תשאול האינדקס

כאשר נרצה לבצע שאילתה על המאגר נחלק את השאילתה ל-2:

1. מציאת הקבצים הרלוונטיים לחיפוש על ידי שימוש באינדקס Needle in a Haystack
2. מציאת התוכן הרלוונטי בתוך קבצי התוכן המקוריים

בחלק זה נתמקד בשלב 1 של תהליך התשאול. בשאילתה על המאגר נקבל שני פרמטרים חשובים:

- קבוצת תנאים המכילה עבור כל תנאי את העמודה עליה נרצה לחפש ואת הערך אותו נרצה לחפש
- אופרטור על גבי התנאים – איחוד או חיתוך בין התנאים

תחילה נבצע תשאול עבור כל תנאי בנפרד באופן הבא:

1. <u>חיפוש קובץ האינדקס המתאימים לערך בעמודה:</u>
   על מנת לבצע את החיפוש נרצה תחילה למצוא את קבצי האינדקס הרלוונטיים לחיפוש שלנו. לשם כך ניעזר בקובץ root index על מנת למצוא את קבצי האינדקס בהם טווח הערכים הקיים מכיל את הערך אותו נחפש.
2. <u>חיפוש הערך בתוך קובץ האינדקס:</u>
   a. לאחר שמצאנו את קובץ האינדקס הרלוונטי נתחיל בחיפוש הערך בתוך ה-bloom filter שלו ונבדוק אם הערך קיים, אם התשובה היא לא אז נוכל להחזיר תשובה שהערך לא קיים במאגר. אם התשובה שתחזור מחיפוש ב-

bloom filter היא שהערך קיים בקובץ אז נמשיך לחיפוש בתוך קובץ האינדקס של הערך.

b. ניגש לאזור ה-metadata של האינדקס ונחפש את החלק בקובץ המכיל את טווח הערכים המכיל את הערך הרצוי.
c. לאחר שמצאנו את החלק הנכון בקובץ האינדקס נחפש בתוכו שורה אחר שורה את הערך הרצוי. אם נמצא את הערך אז נוכל להחזיר את רשימת הקבצים בהם הוא קיים. אם לא אז נחזיר שהערך לא קיים במאגר

לאחר שקיבלנו עבור כל תנאי את רשימת הקבצים בהם הוא קיים, נפעיל את האופרטור אותו קיבלנו כחלק מהשאילתה. לפי האופרטור נבצע איחוד או חיתוך על רשימות הקבצים ונחזיר את רשימת הקבצים לאחר האופרטור ואת הערכים שמצאנו בתוך המאגר.

סיבוכיות זמן התשאול על האינדקס מחולקת באופן הבא:

6. חיפוש בעזרת ה-root index: O(n) כאשר n הוא מספר הערכים במאגר בכל העמודות. קובץ ה-root index הוא ייצוג מוקטן של הערכים אך עדיין בסדר גודל לינארי ומאחר ובמקרה הגרוע ביותר נצטרך לעבור על הקובץ בשלמותו אזי שנעבור על n*c ערכים כאשר c הוא מספר קבוע בין 0 ל-1 המייצג את כמות השורות שניתן לשמור בקובץ אינדקס בודד. ניתן לבצע הרחבה לטובת שיפור עתידי על ידי יצירת חלק של metadata בסוף הקובץ של root index בדומה לקובץ האינדקס הרגיל ובכך להקטין את סיבוכיות הזמן להיות O(m*k) כאשר m הוא מספר העמודות ו-k הוא מספר הערכים המקסימלי בעמודה במאגר.
בעזרת השימוש במנגנון caching עבור ה-root index (כפי שמפורט בהמשך) כאשר רשומת האינדקס מקובץ root index כבר קיימת ב-cache סיבוכיות הזמן של חיפוש רשומה ב-root index לוקחת O(log(k)) כאשר k הוא מספר הערכים המקסימלי בעמודה במאגר.
7. חיפוש בעזרת Bloom Filter: O(1) מאחר ומדובר בבדיקה של מספר סופי של ערכים במטריצת הביטים של מבנה הנתונים.
8. חיפוש בתוך המקטע הרלוונטי בקובץ האינדקס: O(1) מאחר ומספר השורות בכל מקבץ חסומות במספר סופי ניתן להגדרה.

לכן לבסוף סיבוכיות הזמן במקרה הגרוע היא O(n) כאשר n הוא מספר הערכים בכל העמודות במאגר, אך בזכות העובדה כי ישנו פקטור משמעותי של דחיסה והקטנה של הייצוג בקובץ ה-root index הביצועים המושגים בעזרתו מאפשרים שימוש טוב ומהיר במאגר.

5.1.5 הרחבות למאמר המקורי

בפרויקט זה בוצעו שתי הרחבות משמעותיות לאינדקס המתואר במאמר [10] על פי ההצעות שנכללו במאמר וזאת על מנת לשפר את הביצועים של ההכנסה ושל התשאול על גבי המאגר. ההרחבות שבוצעו הן:

1. Bloom Filter:
   מונח זה הוזכר רבות בסעיפים קודמים של הסבר על מבנה האינדקס ונסביר אותו כעת. במאמר [1] מוצגת שיטה לייצוג של קבוצת ערכים מסוימת המאפשרת בדיקה האם ערך מסוים שייך לקבוצת הערכים או לא ולקבל תשובה מרווח שגיאה חד-צדדי אפשרי קטן תוך שמירה על גודל אחסון קטן. מבנה הנתונים מיוצג על ידי מטריצה בגודל קבוע המכילה ביטים של 0 או 1 המייצגים ביחד שייכות של ערכים לקבוצה. כאשר חוזרת תשובה שהערך לא קיים נדע זאת בוודאות וכאשר תחזור תשובה שהערך קיים נדע זאת בהסתברות מסוימת הניתנת לקביעה לפי גודל מבנה הנתונים שבחרנו. מבנה הנתונים מסוגל להגיע לשגיאה של 1 אחוז עם 10 ביטים לכל אובייקט בקבוצה, ללא קשר לגודל האובייקט, משמע חסכוני במקום.
   בפרויקט זה בוצע שימוש במבנה זה עבור כל קובץ אינדקס על מנת לייצג האם ערך מסוים קיים בו או לא. שימוש במבנה נתונים זה עבור קבצי האינדקס מאפשר בזמן חיפוש לבדוק במהירות האם ערך מסוים קיים בקובץ או לא ובכך להחזיר תשובה מהירה יותר לחיפוש של ערכים שאינם קיימים ולחסוך גישה.
   שימוש במבנה נתונים זה צריך להיות מחושב מאחר ובאינדקס התוכן בפרויקט נרצה לחסוך בעלות האחסון כך שהוספת עוד מידע לאינדקס הינה בניגוד לאינטרס. עם זאת, לאור העובדה כי מבנה נתונים זה הינו בעל גודל אחסון נמוך הערך שמתקבל מהשימוש בו עולה על העלות השולית של הוספתו לאחסון.
2. Caching:
   בכל הכנסה לאינדקס ובכל קריאה ממנו ישנו שימוש בקובץ ה-root index המשמש מיפוי בין טווחי הערכים במאגר לבין קבצי האינדקס הרלוונטיים. ככל שמספר הקבצים במאגר גדל כך גם גדל מספר קבצי האינדקס במאגר וגם קובץ ה- root index עצמו גדל. כתוצאה מכך כל קריאה מקובץ זה הופכת להיות יותר ויותר יקרה בעלות החישוב של גישה לקובץ, דבר שלא נרצה שיקרה כדי שלא ייפגעו הביצועים של התשאול וההכנסה במאגר.
   על מנת להפחית את העלות של הגישה לאינדקס נוסף מנגנון caching לכל המערכת. מנגנון ה-caching מבוסס על טכנולוגיית Redis המאפשרת ארכיטקטורת caching מבוזרת ומשותפת בין מספר instances כך שהמנוע של הכנסה למאגר ומנגנון התשאול יכולים שניהם לגשת ל-cache גם אם הם יושבים על מכונות שונות.

בעזרת cache זה אנו יכולים מהר מאוד לבדוק האם יש קובץ אינדקס מתאים לקבוצת ערכים מסוימת על ידי קריאה אחת ל-cache במקום קריאה של כל קובץ ה-root index בכל אינדקס. העובדה כי לא נדרש לקרוא את כל ה-root index בכל פעם מחדש מורידה משמעותית את זמן התשאול וההכנסה עבור מספר ערכים גבוה מאחר ונחסכות הרבה גישות לדיסק וקריאה ממנו, דבר אשר ידוע כמשאב איטי לעומת cache השמור בזיכרון כמו Redis.

5.2 אינדקס HyperLogLog

5.2.1 רקע

במקרים רבים בהם נרצה לנתח מידע על קבצים ב-data lake אחת השאלות הראשונות שנרצה לענות עליהן היא כמה ערכים ייחודיים יש בשדה מסוים. על מנת לענות על שאלה זו באופן ודאי ללא טעות נדרש לשמור את כל הערכים הייחודיים. שמירה של כל הערכים משמעותה שכפול של הרבה מידע וגם שימוש באחסון רב ביחס למה שנרצה, שכן אינדקס לרוב מיועד לשמש כייצוג מוקטן וחסכוני בזיכרון של הרשומות עצמן.

HyperLogLog הוא מבנה נתונים אשר הוצג ב2007 על ידי מספר חוקרים במאמר בשם HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm [2]. מטרתו של מבנה זה היא לאפשר ספירת פריטים ייחודיים ב-multisets עם מרחב שגיאה אפשרי. בעזרת שימוש נכון במבנה נקבל הערכה של מספר האיברים הייחודיים עבור multisets בגודל מעל 10^9 איברים עם שגיאה סטנדרטית של 2 אחוז על ידי שימוש ב1.5 קילובייט של זיכרון.

עבור data lakes בהם נרצה לחסוך באחסון של האינדקס, מבנה נתונים זה הוא אופטימלי לאור הדיוק שלו ביחס לכמות האחסון המזערית אותה הוא צורך. לאור זאת, בפרויקט זה בוצע שימוש במבנה נתונים זה לטובת שמירת אינדקס על כמות הערכים הייחודיים בכל שדה metadata הקיים על הקבצים השמורים במאגר. אינדקס זה מאפשר לקבל תמונת מצב של סוגי הקבצים במאגר ומידע בסיסי עליהם באופן מהיר וללא עלות אחסון משמעותית.

5.2.2 מבנה האינדקס

מאחר והאינדקס מחולק לפי שדות metadata כך גם הקבצים שלו. אם נביט באיור 4 נוכל לראות כי בחלק של אינדקס ה-metadata לכל סוג אינדקס יש תיקייה משלו. במקרה של אינדקס זה שם התיקייה לסוג אינדקס זה הוא HLL_CARDINALITY.

כל HyperLogLog הינו מאחורי הקלעים מערך של m תאים עם ערך מספרי. עבור כל שדה metadata שנפגוש במאגר ניצור קובץ חדש עם שם השדה ועם סיומת hll. בתוך הקובץ

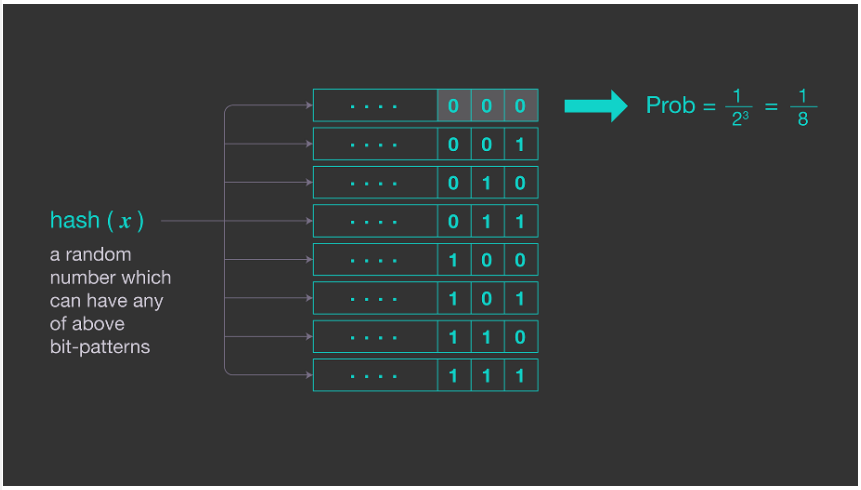
עצמו יישמר ייצוג בינארי של מבנה ה-HyperLogLog עם המערך המייצג שלו ועוד פרטים נוספים הנדרשים לשחזור מדויק של המידע על מבנה הנתונים.

5.2.3 תהליך עדכון האינדקס

כאשר קיים קובץ אינדקס עבור שדה metadata מסוים נטען את הייצוג הבינארי של המידע לזיכרון ועליו נבצע את פעולות ההמשך, אחרת ניצור אובייקט HyperLogLog חדש שיאפשר את הפעולות הנדרשות לעדכון.

על מנת לעדכן את האינדקס עם ערך חדש נבצע את השלבים הבאים:

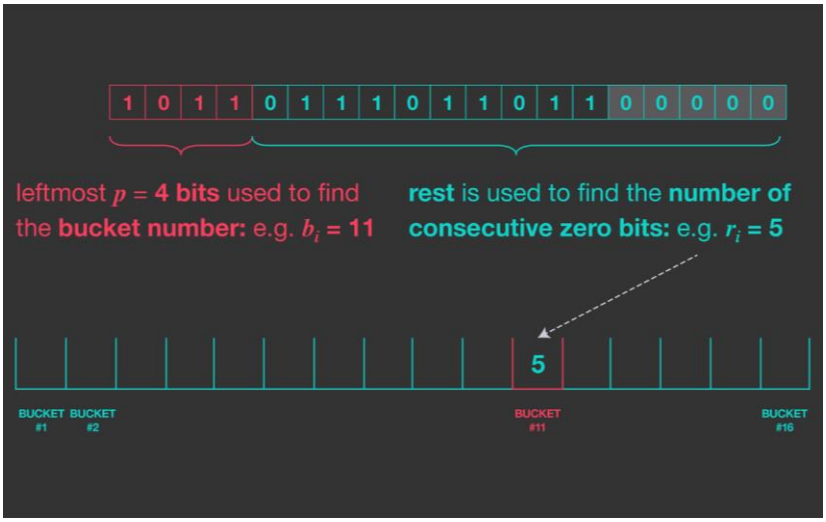
1. ניעזר בעקרון שאומר שבקבוצת ערכים בינאריים ניתן לבצע אומדן למספר הערכים בקבוצה על בסיס מספר האפסים הרצופים הגדול ביותר בייצוג הבינארי של כל אחד מהערכים. עקרון זה מתבסס על העובדה שהסיכוי שיהיו X אפסים ברצף הוא $1/2^x$, כפי שניתן לראות באיור 5. לאור כך אם ישנם X אפסים ברצף במספר מסוים אז ניתן להניח שהיינו צריכים לראות כ-2^x מספרים לפני זה לפי ההסתברות לכך, ולכן ניתן להעריך שבקבוצה יש את 2^x ערכים.



איור 5: המחשה לאומדן כמות המספרים בעזרת מספר האפסים הרצופים בייצוג הבינארי

2. נחשב פונקציית גיבוב על גבי האובייקט אותו נרצה להכניס ונקבל מספר בעל x ביטים.
3. בעזרת k (מספר ניתן להגדרה) הביטים השמאליים ביותר נבחר תא במערך שיצרנו עבור מבנה הHyperLogLog. ניתן לראות דוגמה באיור 6.

4. בעזרת שאר הביטים נמצא את מספר האפסים הרצוף הגדול ביותר ולאחר מכן נשים בתא שבחרנו בשלב 2 את הערך המקסימלי בין מספר האפסים הרצופים שמצאנו בגיבוב הנוכחי לבין המספר הקיים בתא נכון לעכשיו. ערכים אלה ישמשו אותנו בהמשך לטובת התשאול. איור 6 מדגים את אופן הבחירה בעזרת מספר האפסים הרצוף הגדול ביותר.



איור 6: המחשבה לבחירת התא במערך ה-HyperLogLog

סיבוכיות הזמן לעדכון ערך הינה O(1).

5.2.4 תהליך תשאול האינדקס

תשאול האינדקס מתבצע על בסיס שם שדה metadata שנבחר. בעזרת שם השדה הולכים למצוא את הקובץ המתאים המכיל את הייצוג הבינארי של HyperLogLog, וטוענים אותו לזיכרון. לאחר מכן בכדי לקבל קירוב של מספר הערכים הייחודיים בשדה מתבצע התהליך החישובי הבא:

1. ניקח את כל הערכים מהמערך ונוציא את 30 האחוז הגבוהים ביניהם. הסיבה לכך היא כי האלגוריתם נוטה להערכת יתר ולכן נרצה לצמצם את ההשפעה של זה.
2. נחשב את הממוצע ההרמוני על גבי הערכים שנותרו בעזרת הנוסחה הבאה:

$$Z = \left(\sum_{j=1}^{m} 2^{-M[j]} \right)^{-1}$$

$$\alpha_m = \left(m \int_0^\infty \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^m du \right)^{-1}$$

$$E = \alpha_m m^2 Z$$

3. נחזיר את התשובה המתקבלת כתוצאה מהחישוב

החישוב הזה מלווה בשגיאה סטנדרטית של $1.04/\sqrt{m}$ כאשר m הוא גודל המערך שיצרנו.

סיבוכיות זמן הריצה של שליפה על גבי האינדקס היא O(m) כאשר m הוא מספר קבוע שאנחנו מגדירים מראש ביצירת המערך ואינו תלוי במספר הערכים המוכנסים אליו.

5.3 אינדקס Count-Min-Sketch

5.3.1 רקע

במקרים רבים בהם נרצה לנתח מידע על קבצים ב-data lake נרצה לשאול שאלות שקשורות לערך מסוים והימצאותו במאגר, כפי שראינו באינדקס Needle in a Haystack. פרמטר נוסף עליו נרצה להסתכל הוא מספר החזרות של ערך מסוים במאגר, בין אם בתוכן או ב-metadata. על מנת לענות על שאלה זו באופן ודאי ללא טעות נדרש לשמור את כל הערכים הייחודיים ואת מספר המופעים שלהם. שמירה של כל הערכים משמעותה שכפול של הרבה מידע וגם שימוש באחסון רב ביחס למה שנרצה, שכן אינדקס לרוב מיועד לשמש כייצוג מוקטן וחסכוני בזיכרון של הרשומות עצמן.

Count-Min Sketch הוא מבנה נתונים שהוצג ב2009 על ידי Graham Cormode במאמר בשם Count-Min Sketch [3]. מטרת המבנה היא לשמור טבלת תדירות של אירועים או ערכים בתוך קבוצת ערכים מסוימת תוך שימוש בנפח אחסון נמוך. מדובר בהערכה בלבד כאשר האלגוריתם יכול לבצע הערכת יתר אך לעולם לא הערכה בחסר של מספר החזרות של ערך מסוים בתוך קבוצת הערכים. נפח האחסון הוא פחות מנפח לינארי ביחס למספר הערכים.

עבור data lakes בהם נרצה לחסוך באחסון של האינדקס, מבנה נתונים זה הוא אופטימלי לאור הדיוק שלו ביחס לכמות האחסון הקטנה אותה הוא צורך. לאור זאת, בפרויקט זה בוצע שימוש במבנה נתונים זה לטובת שמירת אינדקס על כמות המופעים של ערכים בכל שדה metadata הקיים על הקבצים השמורים במאגר. אינדקס זה מאפשר לקבל תמונת מצב של סוגי הקבצים במאגר ומידע בסיסי עליהם באופן מהיר וללא עלות אחסון משמעותית.

5.3.2 מבנה האינדקס

מאחר והאינדקס מחולק לפי שדות metadata כך גם הקבצים שלו. אם נביט באיור 4 נוכל לראות כי בחלק של אינדקס ה-metadata לכל סוג אינדקס יש תיקייה משלו. במקרה של אינדקס זה שם התיקייה לסוג אינדקס זה הוא CMS_FREQUENCY.

כל Count-Min Sketch הינו מטריצה של m*k תאים עם ערך מספרי בצירוף jל פונקציות גיבוב עם טווח ערכים של 1-m המשמשות לחישוב הערכים במערך. עבור כל שדה

metadata שנפגוש במאגר ניצור קובץ חדש עם שם השדה ועם סיומת cms. בתוך הקובץ עצמו יישמר ייצוג בינארי של מבנה ה-Count-Min Sketch עם המטריצה המייצגת שלו ועוד פרטים נוספים הנדרשים לשחזור מדויק של המידע על מבנה הנתונים.

5.3.3 תהליך עדכון האינדקס

כאשר קיים קובץ אינדקס עבור שדה metadata מסוים נטען את הייצוג הבינארי של המידע לזיכרון ועליו נבצע את פעולות ההמשך, אחרת ניצור אובייקט Count-Min Sketch חדש שיאפשר את הפעולות הנדרשות לעדכון.

נבצע חישוב של j פונקציות הגיבוב על גבי הערך שנרצה להוסיף. עבור תוצאה x של חישוב פונקציית גיבוב מספר i נלך למטריצה המייצגת למיקום ה-x,i ונוסיף לערך הקיים 1. בביצוע החישובים והעדכונים האלה נשמר התיעוד למופע של הערך ב-metadata של קבצי המאגר.

סיבוכיות זמן הריצה של עדכון האינדקס היא O(j) כאשר j הוא מספר פונקציות הגיבוב המוגדרות והוא סופי וניתן להגדרה בהתאם לכמות הערכים אותם נרצה להכניס למבנה הנתונים.

5.3.4 תהליך תשאול האינדקס

תשאול האינדקס מתבצע על בסיס שם שדה metadata שנבחר וערך שנרצה לבדוק את תדירותו. בעזרת שם השדה הולכים למצוא את הקובץ המתאים המכיל את הייצוג הבינארי של Count-Min Sketch, וטוענים אותו לזיכרון. לאחר מכן בכדי לקבל קירוב של מספר המופעים של הערך בשדה מתבצע התהליך החישובי הבא:

1. נבצע חישוב על כל פונקציות הגיבוב של Count-Min Sketch על גבי הערך
2. עבור פונקציה i שבה נקבל תוצאה x נשמור ברשימה את הערך במיקום x,i מהמטריצה של Count-Min Sketch.
3. מבין כל הערכים שמצאנו בשלב 2 ניקח את הערך המינימלי ונחזיר אותו בתור הקירוב למספר המופעים של הערך בשדה ה-metadata. נבחר בערך המינימלי מאחר ובכל תא יכול להיכלל המידע עבור ערך אחד או יותר, כך שהתא בעל "המידע המינימלי", בו נספרו כמה שפחות ערכים, הוא בעל הערך המינימלי והכי קרוב למספר המופעים של הערך במידע שלנו.

[EG4] עם הערות: למה ערך מינימלי? שוב דוגמא להסבר

סיבוכיות זמן הריצה של עדכון האינדקס היא O(j) כאשר j הוא מספר פונקציות הגיבוב המוגדרות והוא סופי וניתן להגדרה בהתאם לכמות הערכים אותם נרצה להכניס למבנה הנתונים. התוצאות המסופקות על ידי האלגוריתם נוטות לבצע הערכת-יתר לאור הצורה בה נשמר המידע במבנה הנתונים.

6 תוצאות

6.1 זמני הכנסה לאינדקס

6.1.1 זמני הכנסה לאינדקס תוכן

Dataset	Number of Files	Total GB	Duration 1	Duration 2	Duration 3	Duration 4	Duration 5	Average Time	Time per GB
CORD	1	0.6130	24.8828	24.3452	24.9025	22.5226	25.1646	24.3635	39.7470
COVID Tracking Project	58	0.0052	1.3156	1.4444	1.3136	1.2728	1.3279	1.3349	257.4553
AWS	15	0.4391	14.0807	14.2860	12.8443	14.0613	15.2215	14.0988	32.1104
Datahub	17	1.1078	59.4995	57.8738	60.3776	54.0435	59.9123	58.3413	52.6646
ECDC	10	0.0853	3.0963	2.8154	2.8523	2.8936	3.3390	2.9993	35.1443
Google	18	6.7391	145.4279	150.6956	155.7309	158.5655	155.4962	153.1832	22.7306
Israeli Health Ministry	22	2.2266	25.7036	26.2407	24.6411	24.0950	27.9866	25.7334	11.5574
Our World in Data	295	0.3206	19.1639	20.4284	18.2455	18.8279	20.4337	19.4199	60.5776

טבלה 2: זמני ריצה של הכנסת datasets לאינדקס התוכן של המאגר

תחילה נסביר על העמודות המוצגות:

- Dataset – שם הdataset שעבד עיבוד בריצה.
- Number of Files – מספר הקבצים שעברו עיבוד בריצה ספציפית עבור dataset.
- Total GB – סך גודל הקבצים (בג'יגה בייט) שעברו עיבוד בריצה ספציפית עבור dataset.
- Duration 1-5: זמני הריצה (בשניות) שנמדדו בחמש ריצות שונות של הכנסת datasetsלתוך האינדקס.
- Average Time – ממוצע זמני הריצה (בשניות) שנמדדו בחמש ריצות שונות של הכנסת datasetsלתוך האינדקס.
- Time per GB – הזמן הממוצע (בשניות) להכנסה לאינדקס של ג'יגה בייט אחד מסוג התוכן שעבר עיבוד בריצה זו.

[EG5] **עם הערות:** גם כאן בשניות?

מתוך הנתונים בטבלה 2 נרצה להסתכל על המגבלות והתנאים להרצת הקבצים במערכת, נקודות הקיצון בנתונים והשוואה של הביצועים שאנו רואים כאן לביצועים שבמאמר שעל בסיסו נכתב הפרויקט.

הפרויקט הורץ בעזרת מכונה וירטואלית בשירותי הEC2 של AWS בעלת מעבד עם 8 ליבות וזיכרון בנפח 32 ג'יגה בייט עם דיסק SSD. ה-resources החיצוניים לפרויקט זה

(Redis, Kafka) הוקמו על גבי תשתית EKS של AWS. לכל אורך הניסוי רץ על המכונה הוירטואלית בכל זמן נתון רק שירות עיבוד המידע והכנסה לאינדקס.

כאשר הפרויקט נכתב נדרשה בחירת איזון בין גודל הזיכרון שיהיה בשימוש על ידי תהליך עיבוד המידע לבין זמני הריצה. היתרון של הגדלת הזיכרון הוא הקטנה של זמן הריצה מאחר ומתבצעת טעינה לזיכרון פעם אחת ולאחר מכן כל פעולות הקלט מתבצעות מול הזיכרון, מה שידוע כמהיר יותר. עם זאת, לנפח זיכרון גבוה יש מחיר כספי משמעותי כאשר בוחרים מכונות להרצת המערכת. לאור המגבלות האלו נבחרה הפשרה הבאה: קבצי המקור ייקראו פעם אחת בלבד מהדיסק על מנת לחסוך בפעולות קלט ויעברו בזיכרון רשומה אחת בכל זמן נתון בכל ליבה עד להגעה לשמירה לאינדקס, אשר יישמר בזיכרון כולו לאור תהליך החישוב שלו. בכך אנו מגבילים את השימוש בזיכרון לגודל האינדקס בלבד, דבר אשר מפחית משמעותית את גודל הזיכרון הנדרש ועם זאת הביצועים לא נפגעו באופן משמעותי כך שהפשרה לא פגעה משמעותית באיכות הפרויקט. הכנסת ה-datasets התבצעה באופן עקבי בין כל ההרצות לפי סדר השורות בטבלה מלמעלה למטה וזאת על מנת לשמור על תנאי קדם זהים.

כאשר מסתכלים על הנתונים בטבלה 2 בעמודת Time per GB ניתן לראות כי יש שתי נקודות קיצון משמעותיות, ה-dataset של COVID Tracking Project וה-dataset של משרד הבריאות הישראלי. הראשון מוסבר לאור גודלו הקטן מה שגרם לכך שזמן הרצת התוכנה וכל התשתית של המערכת נראית משמעותית גדולה יותר יחסית לנפח הקובץ. כתוצאה מכך ניתן לחשוב שהמערכת לא עובדת יעיל עבור סט הקבצים הזה, אך עם זאת אם נסתכל על זמן ההרצאה נראה שמדובר על שנייה אחת בלבד, ולכן ניתן לומר שהמערכת עובדת טוב גם עבור סט קבצים זה. עבור ה-dataset של משרד הבריאות ישנם שני גורמים משמעותיים שהביאו ליעילות. הגורם הראשון הוא מבנה ה-dataset ובעיקר מספר דו ספרתי קטן של קבצים בגודל סביר כך שניתן להריץ אותם באופן מקבילי ויעיל במערכת. הגורם השני הוא תוכן ה-dataset אשר מכיל מעט מאוד ערכים ייחודיים ובכך מפחית את זמן העיבוד הנדרש מאחר ויש סינון ערכים ייחודיים בכניסה לתהליך העיבוד במערכת.

כאשר נשווה את הביצועים המוצגים בטבלה 2 לביצועים של המאמר עליו התבסס הפרויקט [10] נוכל לראות כי מדובר בביצועים דומים בסדרי הגודל של הזמנים ושל נפח האינדקס אל מול נתוני הפתיחה וה-dataset אשר הועלו למערכת. עם זאת חשוב לציין כי בפרויקט זה מומשה גם הצעה מהמאמר לשינוי בעזרת cache ל-root index אך ניכר כי השינוי לא הביא לשיפור משמעותי באזור זה של המערכת.

6.1.2 זמני הכנסה לאינדקס Metadata

Dataset	Number of Files	Total GB	Duration 1	Duration 2	Duration 3	Duration 4	Duration 5	Average Time	Time per File
CORD	1	0.6130	0.1121	0.1194	0.1034	0.1188	0.1215	0.1150	0.1150
COVID Tracking Project	58	0.0052	0.4227	0.4340	0.4607	0.4576	0.3882	0.4327	0.0075
AWS	15	0.4391	0.1750	0.1670	0.1586	0.1645	0.1738	0.1678	0.0112
Datahub	17	1.1078	0.2075	0.1931	0.2190	0.2031	0.2155	0.2076	0.0122
ECDC	10	0.0853	0.1682	0.1658	0.1829	0.1691	0.1847	0.1741	0.0174
Google	18	6.7391	0.2129	0.2202	0.2157	0.2025	0.1977	0.2098	0.0117
Israeli Health Ministry	22	2.2266	0.1980	0.2161	0.1798	0.1892	0.2047	0.1976	0.0090
Our World in Data	295	0.3382	3.6388	3.4513	3.7403	3.6947	3.9622	3.6974	0.0125

טבלה 3: זמני ריצה של הכנסת datasets לאינדקס ה-metadata של המאגר

תחילה נסביר על העמודות המוצגות:

- Dataset – שם הdataset שעבד עיבוד בריצה.
- Number of Files – מספר הקבצים שעברו עיבוד בריצה ספציפית עבור dataset.
- Total GB – סך גודל הקבצים (בג'יגה בייט) שעברו עיבוד בריצה ספציפית עבור dataset.
- Duration 1-5: זמני הריצה שנמדדו בחמש ריצות שונות של הכנסת הdatasets לתוך האינדקס.
- Average Time – ממוצע זמני הריצה (בשניות) שנמדדו בחמש ריצות שונות של הכנסת הdatasets לתוך האינדקס.
- Time per GB – הזמן הממוצע (בשניות) להכנסה לאינדקס של ג'יגה בייט אחד מסוג התוכן שעבר עיבוד בריצה זו.

מתוך הנתונים בטבלה 3 נרצה להסתכל על המגבלות והתנאים להרצת הקבצים במערכת והשוואה של הביצועים שאנו רואים כאן לביצועים שבמאמרים שעל בסיסם נכתב הפרויקט.

הפרויקט הורץ בעזרת מכונה וירטואלית בשירותי הEC2 של AWS בעלת מעבד עם 8 ליבות וזיכרון בנפח 32 ג'יגה בייט עם דיסק SSD. ה-resources החיצוניים לפרויקט זה (Redis, Kafka) הוקמו על גבי תשתית EKS של AWS. לכל אורך הניסוי רץ על המכונה הוירטואלית בכל זמן נתון רק שירות עיבוד המידע והכנסה לאינדקס.

באינדקס זה לא נדרשה פשרה משמעותית באיזון בין זיכרון לביצועים מאחר ונפח הנתונים המעובדים (ה-metadata של הקבצים) הינו זניח ולכן ניתן בקלות לטעון את כולו לזיכרון. עם

זאת היה צורך בהעמקה ולמידה של ספריית Apache Tika על מנת למנוע טעינה של נתונים מיותרים לזיכרון לטובת ניתוח הקבצים, ובכך השימוש בזיכרון למידע מהקבצים עצמם היה קטן מאוד וזניח לעומת שאר רכיבי התוכנה שנטענו לזיכרון על מנת להריץ את המערכת.

הביצועים של האינדקס דומים מאוד למה שהוצג במאמרים [3] [2] עבור זמני הריצה וניתן לראות כי בוצע מימוש יעיל עבורו במערכת הזו.

6.1.3 השוואה בין הזמנים עבור הכנסות לסוגי האינדקסים השונים

לאחר שניתחנו את הביצועים עבור שני סוגי האינדקסים נדרשת גם השוואה בין הנתונים של שני סוגי האינדקס. ניתן לראות בבירור כי אינדקס ה-metadata הינו המהיר והחסכוני בין השניים, באופן מובהק מאוד. מצב זה היה צפוי לאור כמות הנתונים המעובדת עבור האינדקס וכמות הנתונים הנשמרת באינדקס.

עם זאת, ניתן להניח כי אם היה מבוצע שימוש במבנה אינדקס זה לטובת שמירת אינדקס על נתונים דומים עבור תוכן הטבלאות היינו רואים ביצועים משופרים גם כן וזאת לאור המידע הנשמר באינדקס ובזיכרון לטובת צרות אחסון זו.

6.2 גודל אחסון של אינדקס עבור כל ה-Datasets

[EG6] **עם הערות:** עבור איזה DATASET זה?

6.2.1 גודל אחסון עבור אינדקס תוכן

גודל אחסון: 3.8 GB

כמות קבצים: 3527

גודל root index: 622 KB

ניתן לראות כאן שה-root index קטן בסדרי גודל לעומת גודל קבצי האינדקס כולו. המשמעות של הנתון הזה היא שהחיפוש בכדי למצוא את הקובץ הנכון בין קבצי האינדקס הינו מהיר מאוד לאור הגודל הקטן של ה-root index ולאחר מכן כחלק מהמערכת שלנו כל קובץ אינדקס חסום בגודלו (50000 ערכים לקובץ בהגדרות לניסוי).

עם זאת, גודל קבצי האינדקס הינו גדול ביחס לגודל המקורי של קבצי התוכן המקוריים מתוך המאגרים (11 ג'יגה). ניתן למנות לכך מספר סיבות אפשריות:

1. המידע בתוך ה-datasets הינו בעל טווח גדול עם מעט חזרות מה שגורם להרבה רשומות באינדקסים אשר בחלק מהמקרים מגיעות למצב בו אינדקס עבור עמודה מכיל רשומת ערך נפרדת עבור כל ערך בעמודה המקורית כתוצאה מפיזור זה.

2. המערכת כיום שומרת אינדקס עבור כל עמודה במאגר ולא רק עבור עמודות נבחרות, מה שמונע את השליטה על עמודות שהינן יעילות לחישוב אינדקס ואלו שלא.
3. החפיפה בין קבצי המידע שהועלו לאינדקס היא קטנה והאחידות בין שמות העמודות אינה גבוהה וכתוצאה מכך הרבה ערכים נשמרים כפול גם אם הכוונה מאחוריהם הינה זהה.

6.2.2 גודל אחסון עבור אינדקס Metadata

גודל אחסון HyperLogLog: 360 bytes

כמות קבצים HyperLogLog: 10

גודל אחסון Count-Min-Sketch: 11.9 KB

כמות קבצים Count-Min-Sketch: 10

באינדקס זה ניתן לראות ניצול יעיל מאוד של מקום כפי שמובטח על ידי האלגוריתמים כאן. עבור אינדקס ה-HyperLogLog אנחנו רואים את נפח האחסון הקטן בין שני האינדקסים מאחר ומדובר בשמירה הסתברותית של נ0תון מספרי אחד עבור העמודה בניגוד לשמירה הסתברותית של נתון מספרי עבור כל ערך. עם זאת גם באינדקס ה-count min sketch ישנו שימוש מזערי בסדרי גודל לעומת גודל הקבצים אשר ממופים באינדקס. עם זאת חשוב לציין שמדובר במספר קבצי אינדקס נמוך לאור מיעוט בפרטי metadata אותם רצינו לשמור עבור כל קובץ. ניתן להניח שגודל האינדקס יגדל ביחס לינארי לכמות העמודות שנרצה לשמור באינדקס.

6.3 תשאול מבוסס אינדקס

6.3.1 תוצאות תשאול המאגר

Query Type	Average query time	90th percentile query time
NeedleInHaystack	0.1124533333	0.1417194
Count-Min-Sketch	0.02904805333	0.0359478
HyperLogLog	0.02509050667	0.0282432

טבלה 4: סיכום תוצאות הרצה של שאילתות מול מאגר הנתונים

בטבלה 4 ניתן לראות סיכום של תוצאות הרצת השאילתות על מאגר הנתונים דרך ה-API שתואר בפרק 4.2.3. התוצאות הגולמיות במלואן מצורפות בנספח 1. את פירוט תנאי ההרצה והאבחנות שנצפו בזמן ההרצה נפרט בפרקים 6.3.2-6.3.4.

6.3.2 תשאול מבוסס אינדקס תוכן

תכנית ההרצה לתשאול האינדקס עבור תוכן המאגר הורכבה מ3 שאילתות בעלות אופי שונה בכמות התוצאות והערכים באינדקס. ההרצה התבצעה על מכונה וירטואלית בשירותי הEC2 של AWS בעלת מעבד עם 8 ליבות וזיכרון בנפח 32 ג'יגה בייט עם דיסק SSD. ה-resources החיצוניים לפרויקט זה (Redis) הוקמו על גבי תשתית EKS של AWS. לפני תחילת ההרצה הועלו כלל קבצי המאגר כך שניתן יהיה לתשאל עליהם. כחלק מהמימוש של השיפורים המוצעים במאמר [10] קובץ ה-root index נשמר ב-cache עבור תשאול על מנת לייעל את הביצועים.

כפי שניתן לראות התוצאות הינן בעלות ביצועים טובים מאוד (אחוזון 90 מתחת ל150 מילי שניות) כך שניתן לראות את תרומת מנגנון ה-cache לביצועים של התשאול לעומת המאמר המקורי [10].

השאילתות שבוצעו:

1. ה-route אליו נפנה ב-Query API: /api/v1/Query?queryType= NeedleInHaystack
2. תוכן הבקשות
   a. בקשה 1

```
{
    "Relation": "And",
    "Conditions": [
        {
            "ColumnName": "arxiv_id",
            "Value": "0709.0406"
        }
    ]
}
```

   b. בקשה 2

```
{
    "Relation": "And",
    "Conditions": [
        {
            "ColumnName": "date",
            "Value": "2020-01-14"
        }
    ]
}
```

   c. בקשה 3

```
{
    "Relation": "And",
    "Conditions": [
        {
```

```
            "ColumnName": "date",
            "Value": "2021-01-01"
        }
    ]
}
```

6.3.3 תשאול מבוסס אינדקס Metadata

[EG7] עם הערות: דוגמאות לשאילתות כאן?

תכנית ההרצה לתשאול האינדקס עבור ה-metadata הורכבה מ2 שאילתות בעלות שימוש שונה בסוגי האינדקס – אחת השתמשה באינדקס ה-HyperLogLog והשנייה באינדקס ה-Count-Min-Sketch. ההרצה התבצעה על מכונה וירטואלית בשירותי הEC2 של AWS בעלת מעבד עם 8 ליבות וזיכרון בנפח 32 ג'יגה בייט עם דיסק SSD. לפני תחילת ההרצה הועלו כלל קבצי המאגר כך שניתן יהיה לתשאל עליהם.

הביצועים של אינדקס ה-HyperLogLog הינם מהירים מאוד (אחוזון 90 מתחת ל50 מילי שניות) וזאת הודות לגודל האינדקס הקטן כפי שראינו בפרק 6.2.2. הביצועים של אינדקס ה-Count-Min-Sketch דומים מאוד לביצועים של אינדקס ה-HyperLogLog אך עם זאת גבוהים במעט וזאת בעיקר לאור ההבדל הקטן אך קיים בגדלי האינדקס והמידע המצוי בו כפי בתואר בפרק 6.2.2.

השאילתה עבור אינדקס HyperLogLog:

1. ה-route אליו נפנה ב-Query API: /api/v1/Query?queryType=Cardinality
2. תוכן הבקשה:

```
{
    "MetadataKey": "Content-Encoding"
}
```

בקשת השאילתה עבור אינדקס Count-Min-Sketch:

1. ה-route אליו נפנה ב-Query API: /api/v1/Query?queryType= Frequency
2. תוכן הבקשה:

```
{
    "MetadataKey": "Content-Encoding",
    "MetadataValue": "windows-1252"
}
```

6.3.4 השוואה בין סוגי התשאול

כעת לאחר שביצענו ניתוח של ביצועי התשאול בנפרד נרצה להשוות ביניהם. הנתון המעניין ביותר כאן הוא העובדה שזמני הריצה של אינדקס התוכן המבוסס על מאמר [10] אמנם גבוהים יותר מפי 4 מהביצועים של האינדקסים ההסתברותיים המבוססים על מאמרים [2]

[3] אך בהפרש האבסולוטי מדובר על כ100 מילי שניות. כפי שראינו בפרק 6.2 גודל האינדקס של התוכן הינו גדול מעל פי מיליון מהאינדקסים ההסתברותיים, ועם זאת זמן השליפה עליו גדול רק פי 4 ובנוסף ההפרש האבסולוטי הינו קטן מאוד וזניח ביחס למורכבות השאילתה.

ביצועים אלו של אינדקס התוכן נובעים מהגורמים הבאים:

1. אלגוריתם התשאול משתמש במספר מינימלי של קבצים מתוך האינדקס אשר מפחית מאוד את כמות השימוש בתקשורת רשת, דבר אשר מהווה נתח ניכר מזמני התשאול עצמם.
2. הגדרות הריצה של הכנסת המידע לאינדקס התוכן קבעו מגבלה של 50000 ערכים לכל קובץ אינדקס, ובכך הוגבל זמן הריצה על קובץ בודד בקריאה מהאינדקס.
3. השימוש ב-cache עבור קובץ ה-root index מוכיח את עצמו כמאוד יעיל בשליפת המידע במקום הורדתו ממאגר הענן.

7 סיכום

7.1 הצעות לשיפורים בהמשך

לאחר הרצת בדיקות הביצועים והבדיקות הפונקציונליות עלו מספר רעיונות לשיפור שניתן לממש כפרויקט המשך לפרויקט זה:

1. בחירת עמודות לשמירה באינדקס כחלק מתשתית אינדקס התוכן. יכולת זאת תאפשר שליטה על גודל המאגר ולוותר על עמודות שאינן מעניינות לשליפה. ברגע שבוחרים לשמור לאינדקס את כל העמודות ישנו גם סיכוי שהאינדקס יתנפח לאור ריבוי ערכים בעמדות שאינם חוזרים על עצמם כפי שראינו במידה חלקית במקרה שהוצג בפרויקט.
2. שמירה לאינדקס של ערכים חלקיים, לדוגמה שמירה של קישור המילה "קורונה" מתוך הערך "וירוס קורונה". להערכתי תוספת זו לא תדרוש זמן רב אך עם זאת תאפשר תשאול עשיר יותר על המאגר בלי הוספה משמעותית של זמן לתהליך ההכנסה לאינדקס או התשאול על גביו.
3. שמירת cache לתוצאות התשאול על מנת לחסוך בפניות למאגר על תוצאות עדכניות שנשלפו בטווח הזמן הקרוב.

7.2 סיום

לסיכום נראה כי מימוש האינדקס על בסיס מאמר [10] בוצע בצורה יעילה ומכיל את כל היכולות הנדרשות כפי שהוצגו. בנוסף, ניכר כי ביצועי האינדקס של מאמר [10] עומדים בקנה מידה זהה לאינדקסים על metadata המבוססים על מאמרים [2] [3], דבר לא מובן מאליו.

בנוסף מימוש זה מוכיח את יכולת השחזור של תוצאות מאמר [10] בטכנולוגיה שונה לחלוטין ממה שבוצע בו, כאשר המאמר התבסס על טכנולוגיית Spark לטובת מימוש מנגנון ההכנסה לאינדקס והתשאול ולעומת זאת הפרויקט המוצג כאן נכתב מאפס כולו בשפת C# על בסיס טכנולוגיית Microsoft .NET ועל בסיס הפסאודו קוד שמוצג במאמר. העובדה כי הטכנולוגיה אינה מהווה חסם למימוש הארכיטקטורה המוצגת במאמר [10] מוכיחה כי המאמר אינו מקושר לטכנולוגיה ספציפית ובעצם מציג רעיון תיאורטי שניתן למימוש בטכנולוגיות רבות.

הפרויקט היה עבורי דרך כניסה והעמקה בעולמות ה-data lake בענן, תוך כתיבה של קוד מותאם לענן ויצירת תשתיות פיתוח והרצה בסביבת הייצור עבור הוכחת הנכונות והשלמות של הפרויקט. בזכות הפרויקט קיבלתי עוד הצצה לתוך עולמות שירותי הענן וגם גיליתי יכולות מתקדמות ששירתו אותי בעבודה היום-יומית שלי.

8 ביבליוגרפיה

[1] Bloom, B. (1970). Space/time trade-offs in hash coding with allowable errors. Commun. ACM, 13, 422-426.

[2] Flajolet, P., Fusy, É., Gandouet, O., & Meunier, F. (2007). HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. Discrete Mathematics & Theoretical Computer Science, 137-156.

[3] Cormode, G. (2009). Count-Min Sketch. Encyclopedia of Database Systems.

[4] Gani, A., Siddiqa, A., Shamshirband, S., & Nasaruddin, F. (2015). A survey on indexing techniques for big data: taxonomy and performance evaluation. Knowledge and Information Systems, 46, 241-284.

[5] Quix, C., Hai, R., & Vatov, I. (2016). GEMMS: A Generic and Extensible Metadata Management System for data lakes. CAiSE Forum.

[6] Zagan, E., & Danubianu, M. (2020). Data lake Approaches: A Survey. 2020 International Conference on Development and Application Systems (DAS), 189-193.

[7] Couto, J., Borges, O.T., Ruiz, D., Marczak, S., & Prikladnicki, R. (2019). A Mapping Study about data lakes: An Improved Definition and Possible Architectures. SEKE 2019: 453-578.

[8] Derakhshannia, M., Gervet, C., Hajj-Hassan, H., Laurent, A., & Martin, A. (2020). Data lake Governance: Towards a Systemic and Natural Ecosystem Analogy. Future Internet, 12, 126.

[9] Amazon. 2020. Best Practices for Amazon EMR. Retrieved November 14, 2020 from https://d0.awsstatic.com/whitepapers/aws-amazon-emr-best-practices.pdf

[10] Weintraub, G., Gudes, E., & Dolev, S. (2021). Needle in a haystack queries in cloud data lakes. EDBT/ICDT Workshops.

[11] Tan, J., Ghanem, T.M., Perron, M., Yu, X., Stonebraker, M., DeWitt, D.J., Serafini, M., Aboulnaga, A., & Kraska, T. (2019). Choosing A Cloud DBMS: Architectures and Tradeoffs. Proc. VLDB Endow., 12, 2170-2182.

[12] Microsoft. 2020. Azure subscription and service limits, quotas, and constraints. Retrieved November 14, 2020 from

https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/azure-subscription-service-limits

[EG8] **עם הערות**: הסבר קצר על מה רואים בכל טבלה?

9 נספח 1 – תוצאות גולמיות של תשאול המאגר

בטבלה מטה ניתן לראות את התוצאות הגולמיות של זמני ההרצה של השאילתות המתוארות בפרק 6.3 על המאגר שלנו. התוצאות מופרדות לפי סוג האינדקס.

query_type	duration	Num_of_results
HyperLogLog	0.079904	1
HyperLogLog	0.016844	1
HyperLogLog	0.027067	1
HyperLogLog	0.025946	1
HyperLogLog	0.026698	1
HyperLogLog	0.025475	1
HyperLogLog	0.026198	1
HyperLogLog	0.025285	1
HyperLogLog	0.027396	1
HyperLogLog	0.024089	1
HyperLogLog	0.024076	1
HyperLogLog	0.026599	1
HyperLogLog	0.018544	1
HyperLogLog	0.02724	1
HyperLogLog	0.014594	1
HyperLogLog	0.022874	1
HyperLogLog	0.021601	1
HyperLogLog	0.017286	1
HyperLogLog	0.026316	1
HyperLogLog	0.016296	1
HyperLogLog	0.043961	1
HyperLogLog	0.024079	1
HyperLogLog	0.026194	1
HyperLogLog	0.017763	1
HyperLogLog	0.023504	1
HyperLogLog	0.02559	1
HyperLogLog	0.015975	1
HyperLogLog	0.015341	1
HyperLogLog	0.024293	1
HyperLogLog	0.028689	1

HyperLogLog	0.025851	1
HyperLogLog	0.016915	1
HyperLogLog	0.027963	1
HyperLogLog	0.024569	1
HyperLogLog	0.01529	1
HyperLogLog	0.017116	1
HyperLogLog	0.016036	1
HyperLogLog	0.018807	1
HyperLogLog	0.025147	1
HyperLogLog	0.02253	1
HyperLogLog	0.016977	1
HyperLogLog	0.019121	1
HyperLogLog	0.027972	1
HyperLogLog	0.017609	1
HyperLogLog	0.024353	1
HyperLogLog	0.068776	1
HyperLogLog	0.019678	1
HyperLogLog	0.025723	1
HyperLogLog	0.026653	1
HyperLogLog	0.025981	1
HyperLogLog	0.028424	1
HyperLogLog	0.024115	1
HyperLogLog	0.029747	1
HyperLogLog	0.023021	1
HyperLogLog	0.024243	1
HyperLogLog	0.018975	1
HyperLogLog	0.025202	1
HyperLogLog	0.033368	1
HyperLogLog	0.023981	1
HyperLogLog	0.023888	1
HyperLogLog	0.025873	1
HyperLogLog	0.04016	1
HyperLogLog	0.017621	1
HyperLogLog	0.02207	1
HyperLogLog	0.024135	1

HyperLogLog	0.026458	1
HyperLogLog	0.027227	1
HyperLogLog	0.027017	1
HyperLogLog	0.026686	1
HyperLogLog	0.025191	1
HyperLogLog	0.024713	1
HyperLogLog	0.026347	1
HyperLogLog	0.021684	1
HyperLogLog	0.017718	1
HyperLogLog	0.02514	1
Count-Min-Sketch	0.06914	1
Count-Min-Sketch	0.043276	1
Count-Min-Sketch	0.03552	1
Count-Min-Sketch	0.043812	1
Count-Min-Sketch	0.031336	1
Count-Min-Sketch	0.029871	1
Count-Min-Sketch	0.027609	1
Count-Min-Sketch	0.018074	1
Count-Min-Sketch	0.034736	1
Count-Min-Sketch	0.031921	1
Count-Min-Sketch	0.02808	1
Count-Min-Sketch	0.025119	1
Count-Min-Sketch	0.030524	1
Count-Min-Sketch	0.019	1
Count-Min-Sketch	0.030485	1
Count-Min-Sketch	0.021285	1
Count-Min-Sketch	0.021514	1
Count-Min-Sketch	0.025845	1
Count-Min-Sketch	0.019335	1
Count-Min-Sketch	0.039381	1
Count-Min-Sketch	0.028017	1
Count-Min-Sketch	0.018366	1
Count-Min-Sketch	0.026718	1
Count-Min-Sketch	0.040009	1
Count-Min-Sketch	0.024935	1

Count-Min-Sketch	0.025207	1
Count-Min-Sketch	0.020698	1
Count-Min-Sketch	0.019087	1
Count-Min-Sketch	0.029589	1
Count-Min-Sketch	0.030483	1
Count-Min-Sketch	0.019634	1
Count-Min-Sketch	0.026811	1
Count-Min-Sketch	0.034869	1
Count-Min-Sketch	0.030247	1
Count-Min-Sketch	0.02969	1
Count-Min-Sketch	0.031327	1
Count-Min-Sketch	0.029041	1
Count-Min-Sketch	0.023936	1
Count-Min-Sketch	0.026167	1
Count-Min-Sketch	0.023185	1
Count-Min-Sketch	0.019068	1
Count-Min-Sketch	0.027689	1
Count-Min-Sketch	0.036233	1
Count-Min-Sketch	0.027434	1
Count-Min-Sketch	0.025801	1
Count-Min-Sketch	0.031666	1
Count-Min-Sketch	0.029427	1
Count-Min-Sketch	0.064165	1
Count-Min-Sketch	0.028549	1
Count-Min-Sketch	0.028422	1
Count-Min-Sketch	0.021205	1
Count-Min-Sketch	0.0225	1
Count-Min-Sketch	0.020843	1
Count-Min-Sketch	0.028958	1
Count-Min-Sketch	0.027639	1
Count-Min-Sketch	0.019351	1
Count-Min-Sketch	0.027903	1
Count-Min-Sketch	0.027551	1
Count-Min-Sketch	0.021405	1
Count-Min-Sketch	0.031324	1

Count-Min-Sketch	0.024582	1
Count-Min-Sketch	0.027108	1
Count-Min-Sketch	0.022966	1
Count-Min-Sketch	0.0307	1
Count-Min-Sketch	0.024846	1
Count-Min-Sketch	0.03021	1
Count-Min-Sketch	0.029921	1
Count-Min-Sketch	0.060176	1
Count-Min-Sketch	0.030615	1
Count-Min-Sketch	0.028341	1
Count-Min-Sketch	0.023502	1
Count-Min-Sketch	0.020877	1
Count-Min-Sketch	0.03015	1
Count-Min-Sketch	0.034623	1
Count-Min-Sketch	0.028975	1
NeedleInHaystack	2.400628	1
NeedleInHaystack	0.361451	33
NeedleInHaystack	0.268326	116
NeedleInHaystack	0.08794	1
NeedleInHaystack	0.149796	33
NeedleInHaystack	0.130789	116
NeedleInHaystack	0.114647	1
NeedleInHaystack	0.13039	33
NeedleInHaystack	0.122007	116
NeedleInHaystack	0.049983	1
NeedleInHaystack	0.141624	33
NeedleInHaystack	0.135157	116
NeedleInHaystack	0.061745	1
NeedleInHaystack	0.136832	33
NeedleInHaystack	0.141783	116
NeedleInHaystack	0.078814	1
NeedleInHaystack	0.112161	33
NeedleInHaystack	0.113681	116
NeedleInHaystack	0.052488	1
NeedleInHaystack	0.132048	33

NeedleInHaystack	0.129384	116
NeedleInHaystack	0.051052	1
NeedleInHaystack	0.134915	33
NeedleInHaystack	0.11054	116
NeedleInHaystack	0.041279	1
NeedleInHaystack	0.123174	33
NeedleInHaystack	0.142563	116
NeedleInHaystack	0.04967	1
NeedleInHaystack	0.133057	33
NeedleInHaystack	0.161529	116
NeedleInHaystack	0.047613	1
NeedleInHaystack	0.103491	33
NeedleInHaystack	0.123371	116
NeedleInHaystack	0.052191	1
NeedleInHaystack	0.122599	33
NeedleInHaystack	0.141155	116
NeedleInHaystack	0.062218	1
NeedleInHaystack	0.122418	33
NeedleInHaystack	0.107329	116
NeedleInHaystack	0.048219	1
NeedleInHaystack	0.127302	33
NeedleInHaystack	0.112592	116
NeedleInHaystack	0.04802	1
NeedleInHaystack	0.11796	33
NeedleInHaystack	0.1132	116
NeedleInHaystack	0.045886	1
NeedleInHaystack	0.113423	33
NeedleInHaystack	0.120372	116
NeedleInHaystack	0.059171	1
NeedleInHaystack	0.109533	33
NeedleInHaystack	0.15752	116
NeedleInHaystack	0.050304	1
NeedleInHaystack	0.116097	33
NeedleInHaystack	0.117633	116
NeedleInHaystack	0.048512	1

NeedleInHaystack	0.114414	33
NeedleInHaystack	0.116931	116
NeedleInHaystack	0.04541	1
NeedleInHaystack	0.110581	33
NeedleInHaystack	0.111969	116
NeedleInHaystack	0.04312	1
NeedleInHaystack	0.119978	33
NeedleInHaystack	0.106021	116
NeedleInHaystack	0.05161	1
NeedleInHaystack	0.101665	33
NeedleInHaystack	0.151583	116
NeedleInHaystack	0.062422	1
NeedleInHaystack	0.101092	33
NeedleInHaystack	0.170578	116
NeedleInHaystack	0.044258	1
NeedleInHaystack	0.120394	33
NeedleInHaystack	0.163182	116
NeedleInHaystack	0.037344	1
NeedleInHaystack	0.105575	33
NeedleInHaystack	0.129293	116
NeedleInHaystack	0.043913	1
NeedleInHaystack	0.107524	33
NeedleInHaystack	0.108725	116
NeedleInHaystack	0.040439	1
NeedleInHaystack	0.117829	33
NeedleInHaystack	0.102723	116
NeedleInHaystack	0.055713	1
NeedleInHaystack	0.179609	33
NeedleInHaystack	0.115976	116
NeedleInHaystack	0.048794	1
NeedleInHaystack	0.117265	33
NeedleInHaystack	0.128879	116
NeedleInHaystack	0.051214	1
NeedleInHaystack	0.133079	33
NeedleInHaystack	0.110258	116

NeedleInHaystack	0.052308	1
NeedleInHaystack	0.122695	33
NeedleInHaystack	0.137792	116
NeedleInHaystack	0.054859	1
NeedleInHaystack	0.113859	33
NeedleInHaystack	0.151737	116
NeedleInHaystack	0.043174	1
NeedleInHaystack	0.116821	33
NeedleInHaystack	0.120569	116
NeedleInHaystack	0.053838	1
NeedleInHaystack	0.115976	33
NeedleInHaystack	0.127236	116
NeedleInHaystack	0.053997	1
NeedleInHaystack	0.120024	33
NeedleInHaystack	0.112783	116
NeedleInHaystack	0.054211	1
NeedleInHaystack	0.112302	33
NeedleInHaystack	0.126585	116
NeedleInHaystack	0.043531	1
NeedleInHaystack	0.121859	33
NeedleInHaystack	0.139112	116
NeedleInHaystack	0.052504	1
NeedleInHaystack	0.121018	33
NeedleInHaystack	0.145328	116
NeedleInHaystack	0.045274	1
NeedleInHaystack	0.108551	33
NeedleInHaystack	0.112439	116
NeedleInHaystack	0.043292	1
NeedleInHaystack	0.128545	33
NeedleInHaystack	0.116318	116
NeedleInHaystack	0.046001	1
NeedleInHaystack	0.116234	33
NeedleInHaystack	0.110763	116
NeedleInHaystack	0.042375	1
NeedleInHaystack	0.110296	33

NeedleInHaystack	0.109374	116
NeedleInHaystack	0.052992	1
NeedleInHaystack	0.137399	33
NeedleInHaystack	0.110129	116
NeedleInHaystack	0.059916	1
NeedleInHaystack	0.120933	33
NeedleInHaystack	0.125102	116
NeedleInHaystack	0.042113	1
NeedleInHaystack	0.107813	33
NeedleInHaystack	0.114933	116
NeedleInHaystack	0.0513	1
NeedleInHaystack	0.139964	33
NeedleInHaystack	0.11434	116
NeedleInHaystack	0.044986	1
NeedleInHaystack	0.11715	33
NeedleInHaystack	0.11994	116
NeedleInHaystack	0.062198	1
NeedleInHaystack	0.165183	33
NeedleInHaystack	0.132155	116
NeedleInHaystack	0.046327	1
NeedleInHaystack	0.127413	33
NeedleInHaystack	0.177369	116
NeedleInHaystack	0.062874	1
NeedleInHaystack	0.1148	33
NeedleInHaystack	0.103036	116
NeedleInHaystack	0.045527	1
NeedleInHaystack	0.112887	33
NeedleInHaystack	0.131121	116
NeedleInHaystack	0.061113	1
NeedleInHaystack	0.162568	33
NeedleInHaystack	0.112947	116
NeedleInHaystack	0.046574	1
NeedleInHaystack	0.122781	33
NeedleInHaystack	0.127019	116
NeedleInHaystack	0.052473	1

NeedleInHaystack	0.114247	33
NeedleInHaystack	0.109257	116
NeedleInHaystack	0.049347	1
NeedleInHaystack	0.121044	33
NeedleInHaystack	0.144318	116
NeedleInHaystack	0.041195	1
NeedleInHaystack	0.144348	33
NeedleInHaystack	0.116977	116
NeedleInHaystack	0.052332	1
NeedleInHaystack	0.167203	33
NeedleInHaystack	0.112598	116
NeedleInHaystack	0.047282	1
NeedleInHaystack	0.125721	33
NeedleInHaystack	0.115667	116
NeedleInHaystack	0.050973	1
NeedleInHaystack	0.121761	33
NeedleInHaystack	0.142106	116
NeedleInHaystack	0.053438	1
NeedleInHaystack	0.11282	33
NeedleInHaystack	0.115618	116
NeedleInHaystack	0.054962	1
NeedleInHaystack	0.120017	33
NeedleInHaystack	0.110033	116
NeedleInHaystack	0.044487	1
NeedleInHaystack	0.113883	33
NeedleInHaystack	0.127829	116
NeedleInHaystack	0.044238	1
NeedleInHaystack	0.115303	33
NeedleInHaystack	0.100395	116
NeedleInHaystack	0.046612	1
NeedleInHaystack	0.118276	33
NeedleInHaystack	0.116906	116
NeedleInHaystack	0.043719	1
NeedleInHaystack	0.112693	33
NeedleInHaystack	0.110523	116

NeedleInHaystack	0.050089	1
NeedleInHaystack	0.122965	33
NeedleInHaystack	0.126849	116
NeedleInHaystack	0.067919	1
NeedleInHaystack	0.123355	33
NeedleInHaystack	0.191801	116
NeedleInHaystack	0.048275	1
NeedleInHaystack	0.123304	33
NeedleInHaystack	0.146072	116
NeedleInHaystack	0.043942	1
NeedleInHaystack	0.124779	33
NeedleInHaystack	0.125006	116
NeedleInHaystack	0.060357	1
NeedleInHaystack	0.118513	33
NeedleInHaystack	0.115704	116
NeedleInHaystack	0.05802	1
NeedleInHaystack	0.126702	33
NeedleInHaystack	0.111923	116
NeedleInHaystack	0.043687	1
NeedleInHaystack	0.112751	33
NeedleInHaystack	0.119887	116
NeedleInHaystack	0.055962	1
NeedleInHaystack	0.129654	33
NeedleInHaystack	0.133535	116
NeedleInHaystack	0.070535	1
NeedleInHaystack	0.12813	33
NeedleInHaystack	0.121317	116
NeedleInHaystack	0.055592	1
NeedleInHaystack	0.115903	33
NeedleInHaystack	0.105447	116

10 נספח 2 – דוגמה לקבצי אינדקס לפני ואחרי עדכון

10.1 לפני עדכון

10.1.1 קובץ Root Index

{"FileName":"__INDEX__/__COLUMNS__/Column/1.txt","Max":"2","Min":"1","
ColumnName":"Column"}

10.1.2 קובץ אינדקס לעמודה Column

{"Files":["1.json"],"Value":"1"}
{"Files":["2.json"],"Value":"2"}
{"Offset":0,"Max":"2","Min":"1"}
binary representation of bloom filter
offset for the beginning of the metadata section
offset for the beginning of the binary bloom filter section

10.2 תוכן העדכון

שם הקובץ: 3.json

תוכן הקובץ:

Column
3

10.3 לפני עדכון

10.3.1 קובץ Root Index

{"FileName":"__INDEX__/__COLUMNS__/Column/1.txt","Max":"3","Min":"1","
ColumnName":"Column"}

10.3.2 קובץ אינדקס לעמודה Column

{"Files":["1.json"],"Value":"1"}
{"Files":["2.json"],"Value":"2"}
{"Files":["3.json"],"Value":"3"}
{"Offset":0,"Max":"3","Min":"1"}
binary representation of bloom filter

offset for the beginning of the metadata section

offset for the beginning of the binary bloom filter section