האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב
החטיבה למדעי המחשב

פרויקט מתקדם במדעי המחשב:

מימוש פעולת צימוד של MapReduce ב-NoSQL DB

מנחה: פרופ' אהוד גודס

סמסטר ב' 2014

מגיש:
בורוביצקי אלכסנדר, ת.ז. 314542598

פרויקט מתקדם זה הוגש כחלק מהדרישות לקבלת תואר מוסמך (M.Sc.) במדעי המחשב באוניברסיטה הפתוחה
פברואר 2014

תוכן עניינים

תקציר

פעולת הצימוד (Join) היא פעולה מרכזית בבסיסי נתונים. בסביבת הענן (Cloud) יש צורך לבצע את צימוד על בסיסי נתונים מאוד גדולים, לפיכך פותחו כמה אלגוריתמים לחישוב מקבילי ויעיל של צימוד בסביבת Map-Reduce Framework שמוסבר בסעיף 2.2.

בפרויקט זה אנו מממשים את אלגוריתם של ג'פרי אולמן [1] עבור פעולת צימוד (Join) (שפותח עבור מודל ה-MapReduce) בבסיס נתונים ה-NoSQL.

בסעיף 2.1 אנו מתארים את מערכת בסיסי נתונים ה-NoSQL. בסעיף 2.3 אנו מציגים את בסיס נתונים ה-NoSQL הנבחר בפרויקט זה. בסעיף 3 אנו מגדירים את בסיס נתונים כאשר היחסים מחוברים בינהם בצורת כוכב (Star). סעיף 4 מתאר את מימוש הפרויקט כולל הגדרת המחלקות, גישה לנתונים, טעינת נתונים וביצוע של פעולת צימוד לפני ואחרי מימוש האלגוריתם של אולמן. בסוף הסעיף אנו משווים את הביצועים בין שתי גישות.

1. מטרת הפרויקט

מטרת הפרויקט היא לממש את אלגוריתם של ג'פרי אולמן [1] עבור פעולת צימוד (Join) (שפותח עבור מודל ה-MapReduce) בבסיס נתונים ה-NoSQL. האלגוריתם מייעל באופן משמעותי את פעולת צימוד בין יותר מ-2 יחסים ב-Map-Reduce Framework.

אפשר גם לממש את האלגוריתם מחוץ ל-Map-Reduce.

בעיקרון אפשר לממש את אלגוריתם של אולמן בכל NoSQL DB. החלק הקשה זה למצוע את NoSQL DB אשר ייתן אפשרות להחליף את פונקציונליות מקורית של פעולת צימוד.

ולכן ה-NoSQL DB הנבחר הוא Berkeley DB Java Edition [3] אשר מכיל את ממשק עבור פונקציונליות כזאת (ה-API) וישנה אפשרות להחליף את האלגוריתם המקורי של הממשק.

בהמשך נציג בקצרה את עיקרון של NoSQL DB, וגם נציג את ה- Map-Reduce Framework. בנוסף גם נראה את המבנה של Berkeley DB ומבנה הפרויקט עצמו.

2. מבוא

אחת הפעולות המשותפות בביצוע שאילתות גם ב- DBMS וגם ב-Map-Reduce היא פעולת צימוד. פעולת צימוד מאחדת את 2 או יותר יחסים בדרך כלל על סמך תנאי מסוים. פעולת צימוד נחקרה לעומק, וישנם מספר אלגוריתמים שמטפלים בפעולה הזאת. Nested-Loops Join, Sort-Merge Join ו-Hash Join אלא דוגמאות לאלגוריתמים פופולרים עבור פעולת צימוד. כאשר אוסף נתונים לביצוע צימוד הוא גדול מאוד האלגוריתמים האלה הופכים ללא יעילים. לכן לצורך תמיכה יעילה יותר פותחו מודלים חדשים. אחד המודלים האלה שמציגים בסעיף 2.1 הוא מערכת בסיסי נתונים ה-NoSQL.

2.1 מערכת בסיסי נתונים ה-NoSQL [2]

בסיס נתונים ה- NoSQL הוא סוג של מערכת ניהול מאגרי נתונים לא יחסיים המיועדת לבסיסי נתונים מבוזרים כאשר יש צורך בשמירת כמות נתונים גדולה

מאוד. סוגים כאלה של מערכות לא חייבים לדרוש סכימה קבועה, ובדרך כלל תגדל "אופקית".
להלן כמה מאפיינים חשובים של מערכת בסיסי נתונים ה-NoSQL:

- ללא סכימה:
  סוגים שונים של מסמכים יכולים להישמר באותו אוסף:

```
{"color" : "red"}
{"salary" : 100.00}
```

- גידול "אופקית":
  לגדול אופקית כוונה להוסיף עוד צמתים למערכת ולחלק עומס בין הצמתים האלה.
- מאפיינים בסיסים:
  - ✓ זמינות.
  - ✓ בסופו של דבר עקבי.
  - ✓ תשובות קרובות מקובלות.
  - ✓ פשוטות ומהירות.

סוגים של בסיסי נתונים ה-NoSQL:

- מסמך: נתונים נשמרים בצורת מסמך.

לדוגמה:

```
FirstName="Arun", Address="St. Xavier's Road",
Spouse=[{Name:"Kiran"}], Children=[{Name:"Rihit", Age:8}]
```

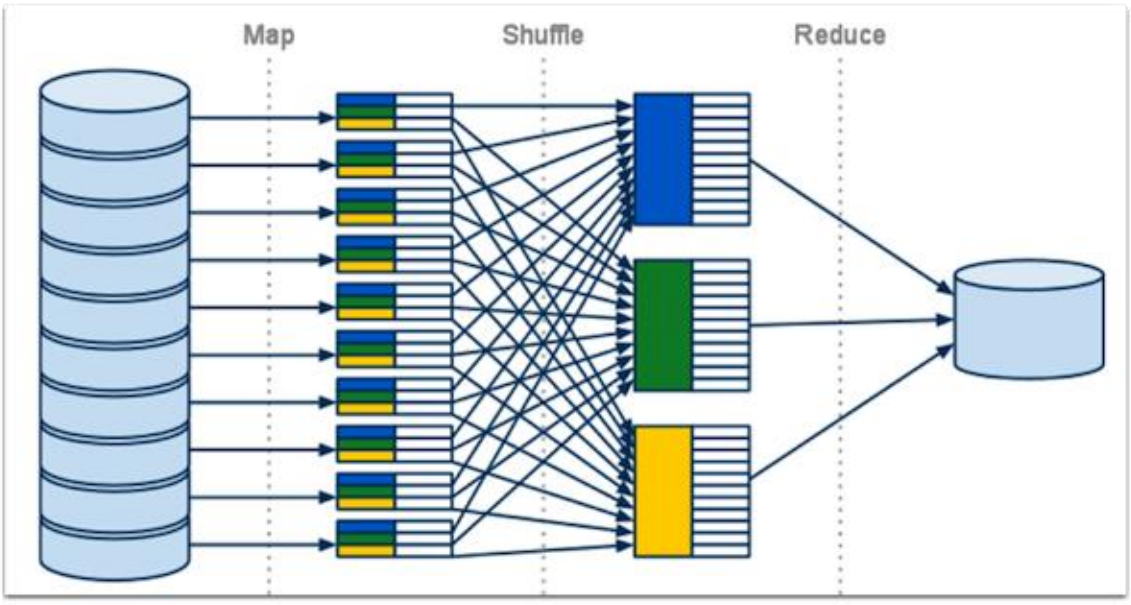
- XML: נתונים נשמרים בפורמט של XML.
- גרף: נתונים נשמרים כי אוסף של צמתים כאשר צומת דומה לאובייקט בשפת תיכנות.
- Key/Value: בסוג זה של בסיס נתונים משתמש יכול לשמור נתונים ללא סכימה. המפתח יכול להיות מסוג String, Hash, List, Set ונתונים נשמרים תחת מפתחות אלה.

הערה: בבסיס נתונים ה-NoSQL הנבחר הנתונים נשמרים בצורת Key/Value.

2.2 [1] Map-Reduce Framework

העיקרון המרכזי מאחורי מודל ה-Map-Reduce הוא עיבוד נתונים במקביל: הרי כיוון שמדובר בכמות נתונים גדולה מאוד, חייבים לחלק נתונים בין אלפי מחשבים כך שהעיבוד יסתיים בזמן סביר.
כמו שאפשר לראות באיור 1 התהליך חישוב כללי ב-Map-Reduce מתבצע בשלבים:
בשלב ה-Map מסווגים את הרשומות לפי מפתח ושולחים את כל הרשומות עם אותו מפתח לצומת ה-Reduce אחד אשר מעבד את הנתונים ומבצע את החישוב. בצורה זו ניתן לבצע במקביל את פעולות חישוביות רבות על מאות או אלפי צמתים.

איור 1 תהליך כללי של Map-Reduce



להלן דוגמה לתהליך החישוב ב-Map-Reduce:
נניח שרוצים לחשב את מספר מופעים של כל מילה מקלט
ב-Map-Reduce . אפשר לתאר את החישוב בעזרת פסבדו קוד הבא:

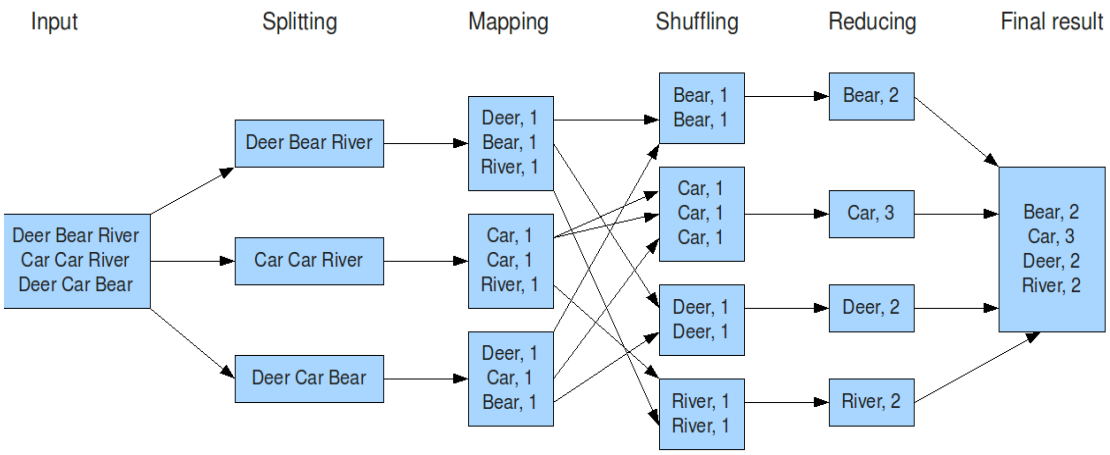
```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value do:
   emit(w, "1");

reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values do:
     result += v;
```

כאשר תפקיד של פונקציית emit הוא למפות את תוצאות לפלט.

איור 2 מדגים את התהליך.

איור 2 תהליך החישוב של מספר מופעים של כל מילה מקלט ב-Map-Reduce



שלב ה-Input: הקלט מתקבל במערכת ונשלח לשלב ה-Splitting.

שלב ה-Splitting: מחלקים את הקלט למילים ושולחים לשלב ה-Mapping.

שלב ה-Mapping: מסווגים את כל מילה
(כרגע מספר המופעים עבור כל מילה הוא 1) ומפעילים את
שלב ה-Shuffling.

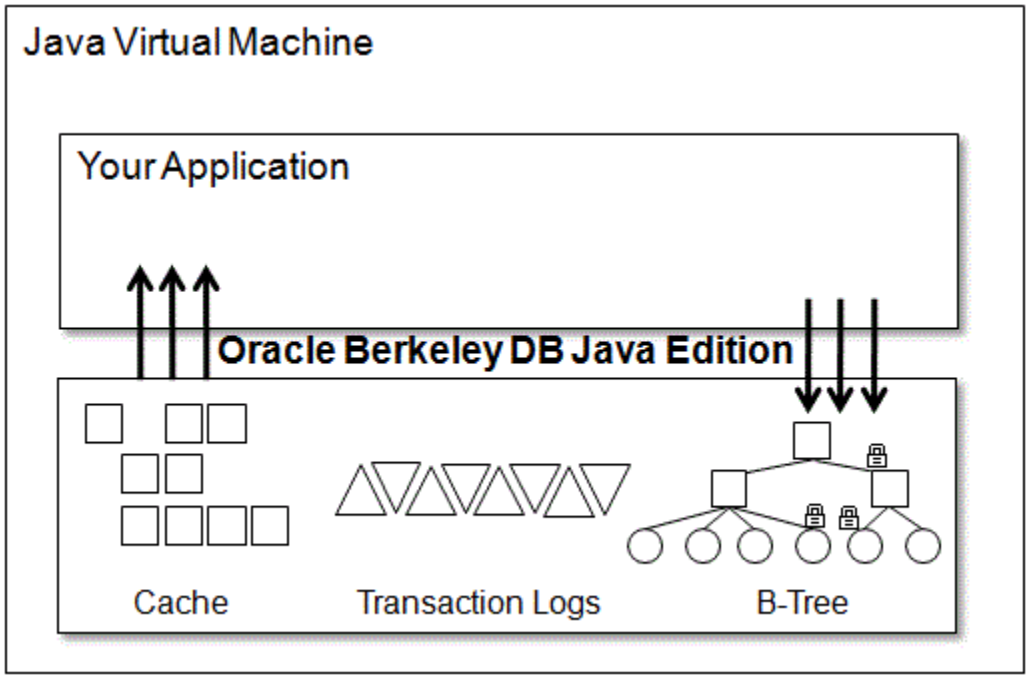
שלב ה-Shuffling: כל צומת של ה-Shuffling מקבל את סיווג ייחודי של מילים,
ומפעיל את שלב ה-Reducing.

שלב ה-Reducing: מחשבים את מספר מופעים של כל מילה ושולחים את
תוצעות לפלט.

2.3 בסיס נתונים Berkeley [3], [4]

בסיס נתונים ה-NoSQL הנבחר הוא Berkeley DB Java Edition שנותן את ממשק (ה-API) לפעולת צימוד ומאפשר להחליף את האלגוריתם המקורי.
המבנה של בסיס נתונים Berkeley מוצג באיור 3.

איור 3 מבנה של בסיס נתונים Berkeley DB Java Edition



בסיס נתונים Berkeley Java Edition כתוב 100% בשפת Java.
ה-Berkeley מאפשר ,בעזרת ממשקים שונים, לקרוא ולכתוב נתונים, לנהל בסיסי נתונים ולנהל טרנזקציות שמתרחשות בבסיס נתונים.

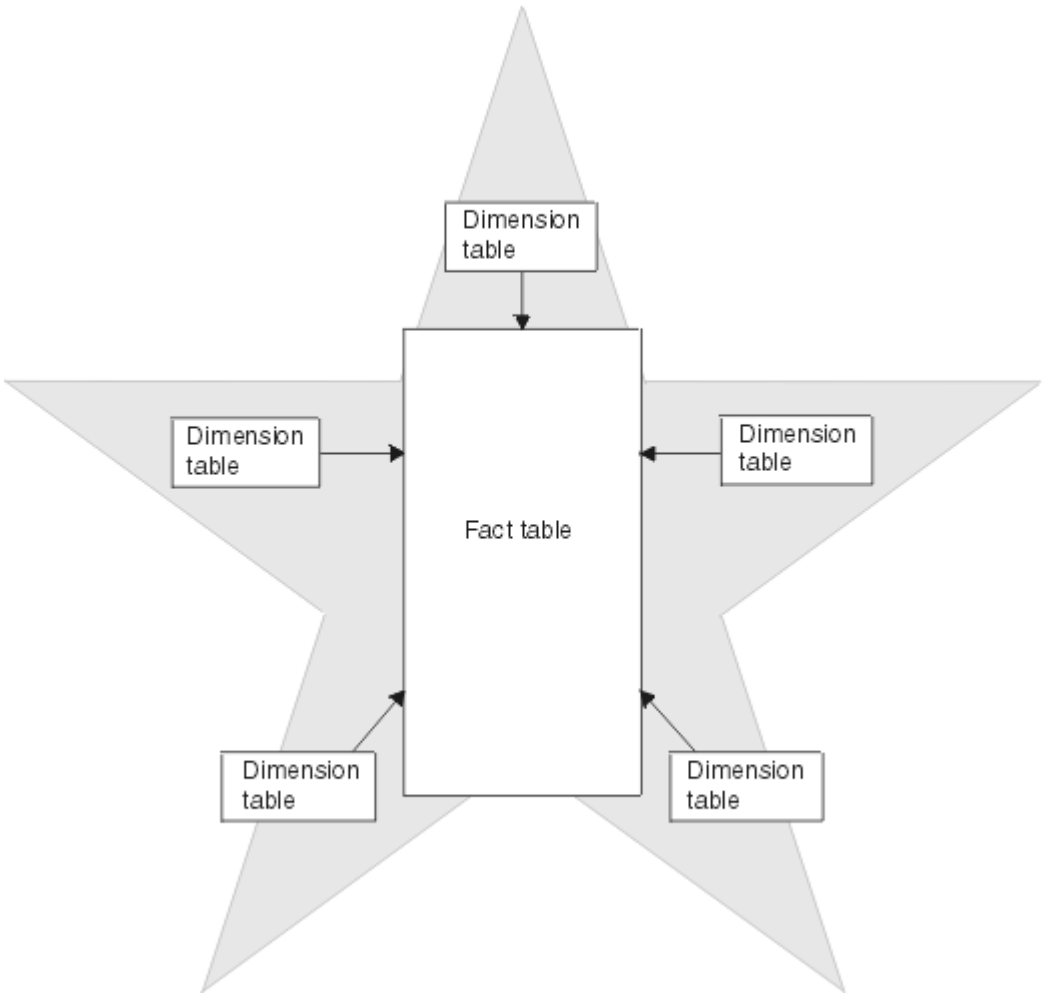
להלן מחלקות החשובות של Berkeley שהשתמשנו במהלך הפרויקט:

- **Environment –** פתיחה וסגירה של בסיס נתונים מתבצעות בעזרת ישות Environment.
- **Entity** – מחלקה זו מייצגת את יחס בבסיס נתונים.
- **PrimaryIndex** – בעזרת מחלקה זו ניגשים לנתונים בבסיס נתונים.

3. בסיס נתונים בצורת כוכב (Star)

בפרויקט זה נשתמש בבסיס נתונים של מכירות (Sales) כאשר יחסים מחוברים ביניהם בצורת כוכב (Star) כפי שמוצג באיור 4.

איור 4 יחסים המחוברים בצורת כוכב



היחס המרכזי (Fact) הוא מכירות (Sales).
היחסים הנלווים הם:

- מוצר (Product)
- לקוח (Customer)
- מיקום העסקה (Location)
- זמו העסקה (Time)

בסעיף 4.7 נציג את אלגוריתם החדש לביצוע של פעולת צימוד בעזרת Map-Reduce.

4. מימוש הפרויקט

בסעיף זה נציג את מימוש הפרויקט

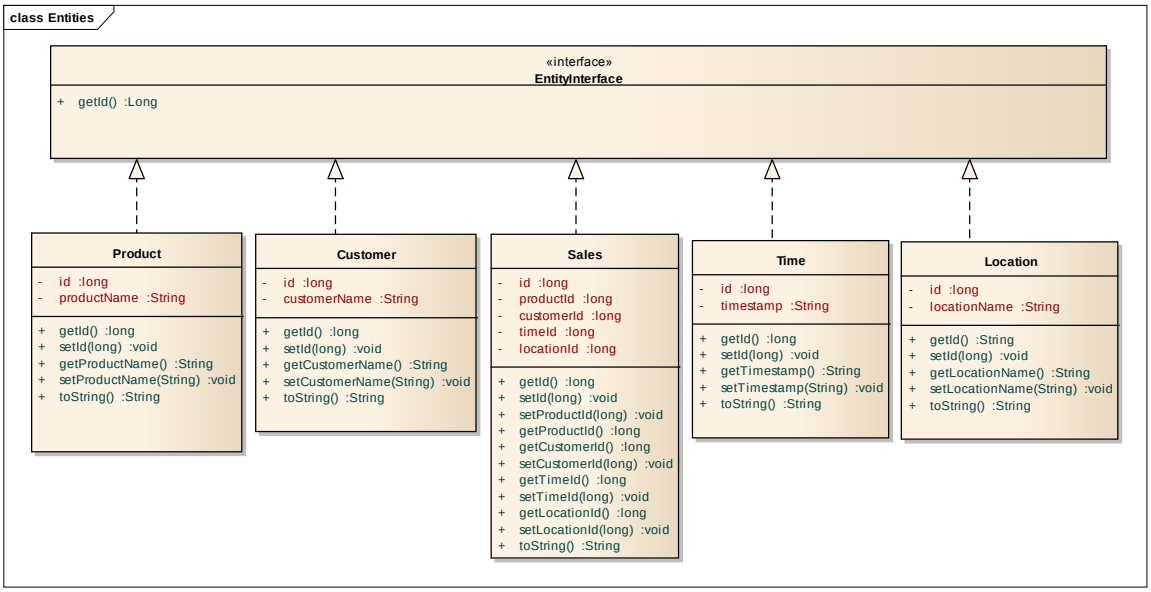
4.1 ישויות

כיון שאנחנו מתעסקים עם בסיס נתונים, צריכים להגדיר ישות שתייצג את יחס.
להלן רשימת ישויות האלה:

- מוצר (Product)
  לישות Product ישנם שדות הבאים:
  1. מזהה (id)
  2. שם המוצר (product name)
- לקוח (Customer)
  לישות Customer ישנם שדות הבאים:
  1. מזהה (id)
  2. שם הלקוח (customer name)
- זמן העסקה (Time)
  לישות Time ישנם שדות הבאים:
  1. מזהה (id)
  2. חותם זמן (timestamp)
- מיקום (Location)
  לישות Location ישנם שדות הבאים:
  1. מזהה (id)
  2. שם המיקום (location name)
- מכירות (Sales)
  לישות Sales ישנם שדות הבאים:
  1. מזהה (id)
  2. מזהה המוצר (product id)
  3. מזהה הלקוח (customer id)
  4. מזהה הזמן העסקה (time id)
  5. מזהה המיקום (location id)

איור 5 מציג את תרשים הישויות.

איור 5 תרשים ישויות



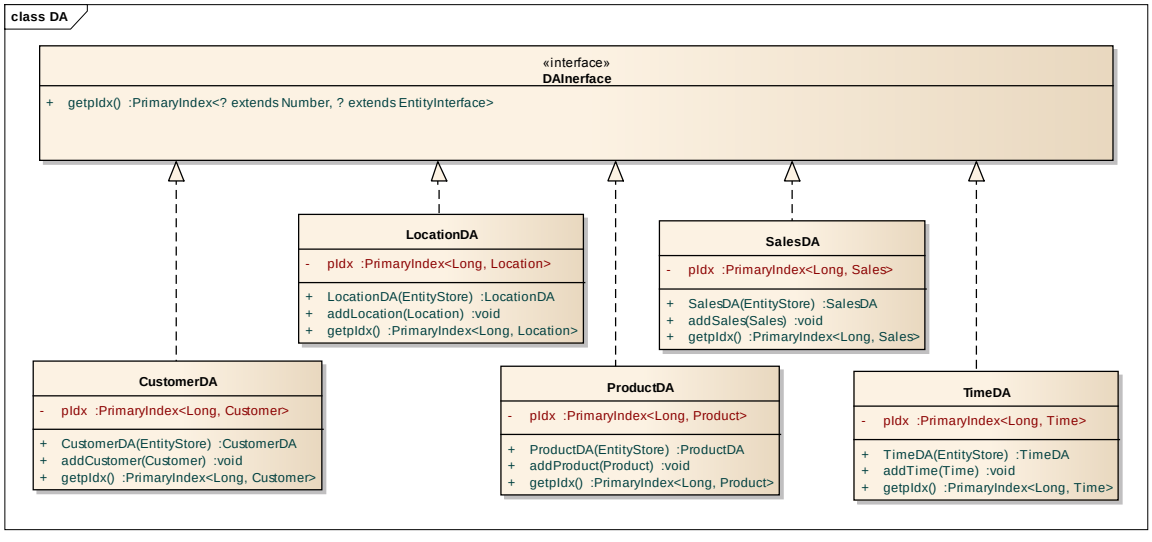
4.2 גישה לנתונים בבסיס נתונים

עבור כל ישות במערכת הגדרנו מחלקות מיוחדות לצורך גישה לנתונים בבסיס הנתונים.
הגישה מתבצעת בעזרת מחלקת PrimaryIndex
להלן רשימת מחלקות המורשות:

- ProductDA – גישה לנתונים של מוצרים.
- CustomerDA – גישה לנתונים של לקוחות.
- TimeDA – גישה לנתונים של זמני העסקאות.
- LocationDA – גישה לנתונים של מיקום.
- SalesDA – גישה לנתונים של מכירות

איור 6 מציג את תרשים המחלקות לצורך גישה לנתונים.

איור 6 תרשים מחלקות לצורך גישה לנתונים

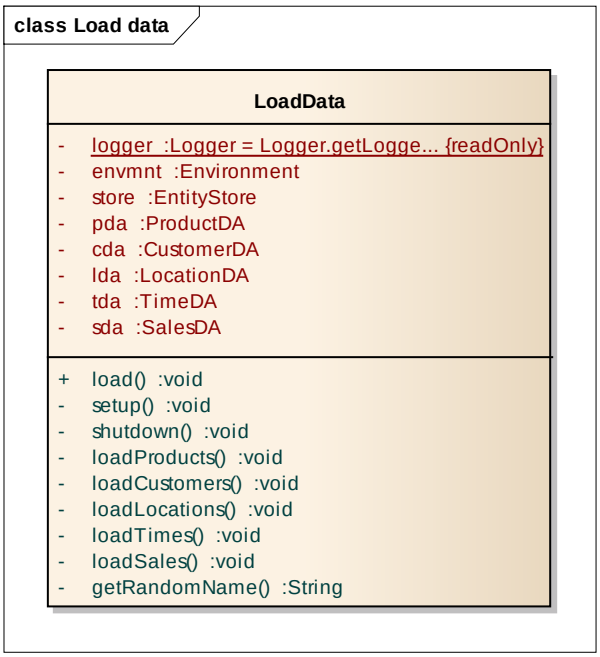


4.3 טעינת נתונים

לצורך טעינת נתונים פיתחנו מחלקה בשם LoadData.

איור 7 מציג את תרשים המחלקה לטעינת נתונים.

איור 7 תרשים המחלקה לטעינת נתונים



4.4 מבנה המחלקה לטעינת נתונים

פעולות הבאות מוגדרות במחלקה (בהמשך נסביר מטרה של כל אחת מפעולות האלה):

1. setup
2. load
3. shutdown
4. loadProducts
5. loadCustomers
6. loadLocations
7. loadTimes
8. loadSales
9. getRandomName

4.5 תהליך הטעינה

הנתונים מיוצרים בצורה סינטטית בזמן הטעינה.
להלן התהליך של טעינת נתונים לבסיס הנתונים:

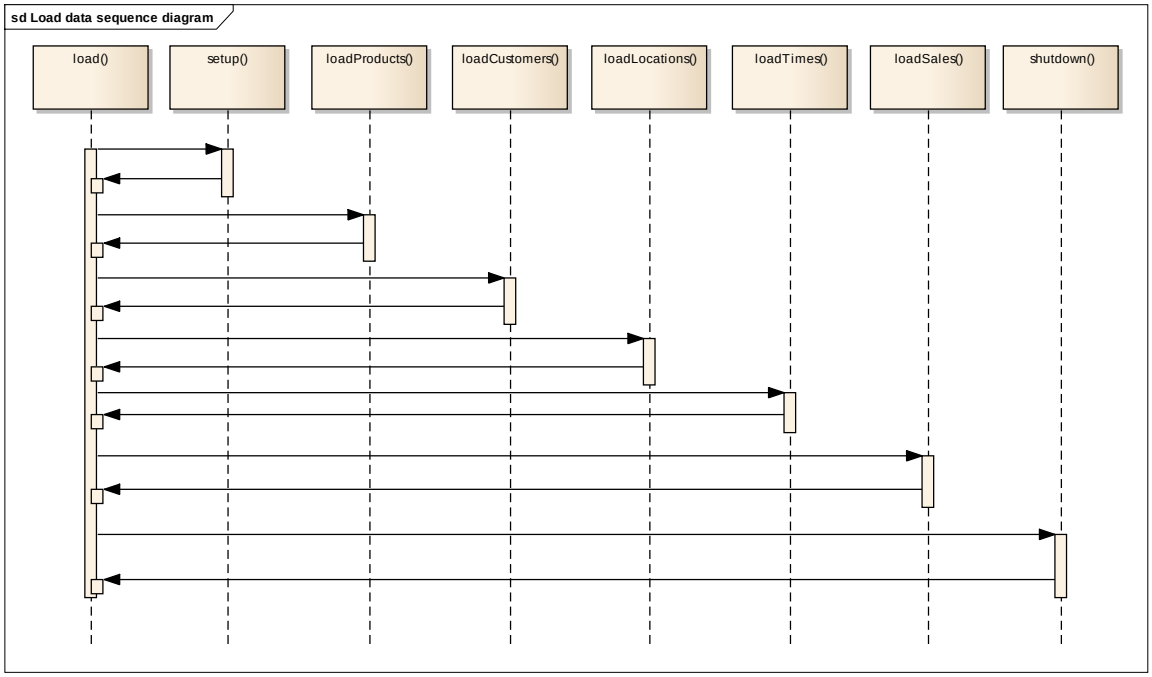
שלב 1 – setup: הכנת סביבת עבודה מול בסיס הנתונים (Environment) כולל חיבור לבסיס הנתונים.

שלב 2 – load: טעינת כל ישות הפרויקט לבסיס הנתונים. בזמן הטעינת כל ישות מייצרים שמות אקראים (פעולת getRandomName) עבור מוצרים, לקוחות ומיקום העסקה שונים. עבור זמן העסקה מייצרים זמן אקראי.

שלב 3 – shutdown: סגירת חיבור לבסיס נתונים

איור 8 מציג תרשים זרימה לטעינת נתונים.

איור 8 תרשים זרימה לטעינת נתונים

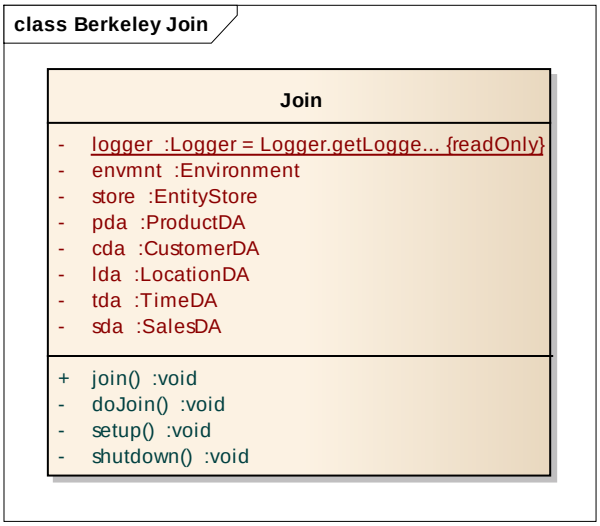


4.6 פעולת צימוד (Join) בבסיס נתונים Berkeley

לצורך ביצוע של פעולת צימוד בבסיס נתונים Berkeley פיתחנו מחלקת Join.

איור 9 מציג את מחלקת Join לביצוע פעולת צימוד בבסיס הנתונים

איור 9 תרשים מחלקת Join

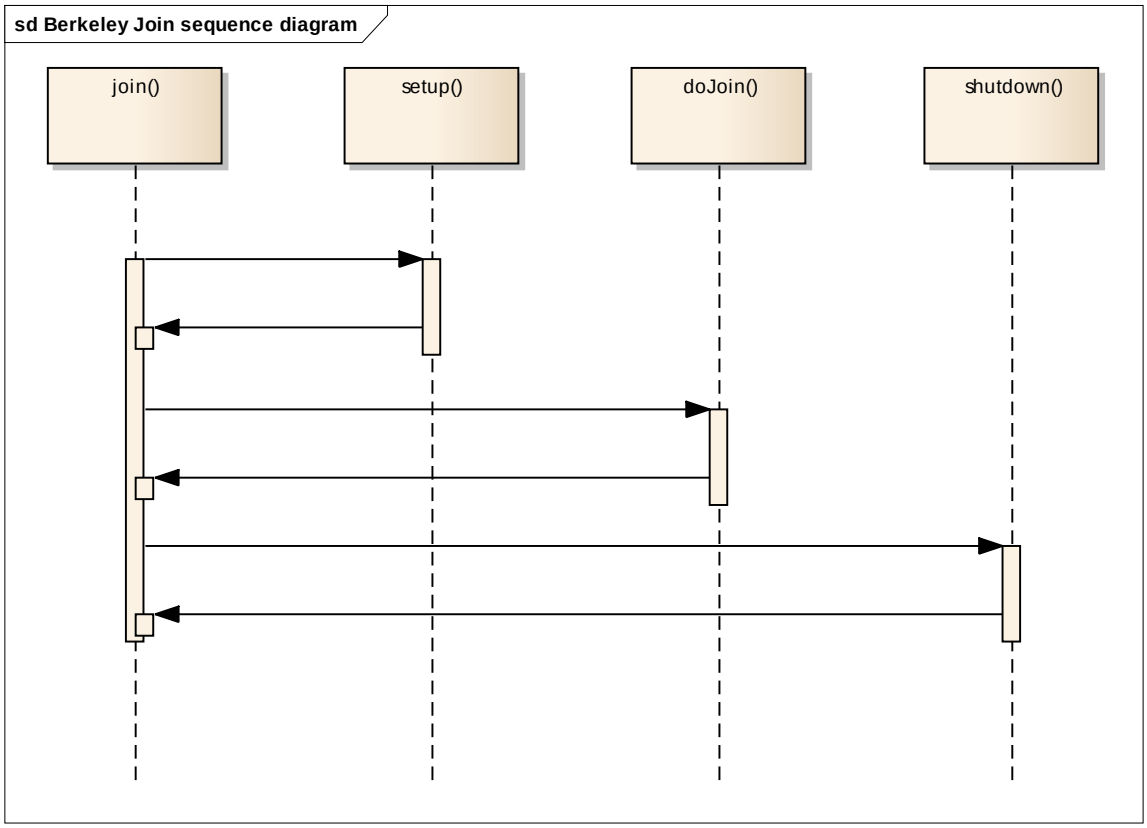


הפעולה האחראית לביצוע של פעולת צימוד היא doJoin.

תהליך ביצוע של הפעולת צימוד מתואר בפסבדו-קוד הבא:

1. קבל כל הרשומות מיחס המרכזי של מכירות.
2. עבור כל רשומה מיחס מכירות בצע:
   a. גש ליחס של מוצרים עם מזהה של המוצר המופיע ברשומה של מכירות, וקבל את רשומה כולה.
   b. עשה זאת עבור יחסים של לקוחות, זמנים ומקומות.

איור 10 מציג תרשים זרימה לביצוע צימוד בבסיס הנתונים.

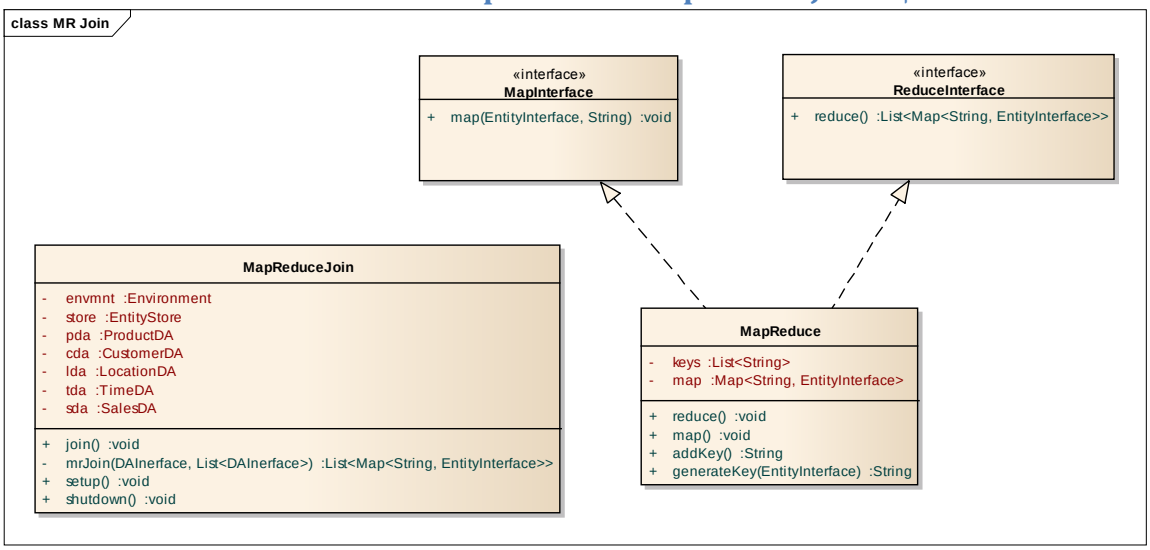


4.7 מימוש פעולת צימוד של Map-Reduce ב-NoSQL

על מנת לממש את פעולת צימוד מ-Map-Reduce בבסיס נתונים של Berkeley השתמשנו באלגוריתם של ג'פרי אולמן המתאור בעבודה המסכמת ומופיע כנספח א'. פיתחנו את 2 מחלקות עבור מימוש זה: MapReduceJoin ו-MapReduce.

איור 11 מציג את מחלקות MapReduceJoin ו-MapReduce.

איור 11 תרשים מחלקות MapReduceJoin ו-MapReduce



הפעולה האחראית לביצוע את פעולת צימוד היא mrJoin. בעצם כאן לא מתבצעת פעולת צימוד במובן של בסיס נתונים: עבור כל ישות בפרויקט מבצעים את שלב המיפוי (Map), ואז בשלב ה-Reduce מבצעים את צימוד על סמך המיפוי: עוברים על כל רשומה של המכירות ומחפשים מיפוי מתאים לכל הרשומות הנלוות על סמך מזהים של המוצר, הלקוח, הזמן והמיקום. כך נוכל לבצע צימוד מלא בין ישות מרכזית וכל הישויות הנלוות.
בסעיף הבא נשווא את ביצועים של פעולת צימוד בין גישה של NoSQL ו-MapReduce.

תהליך הביצוע של הפעולת צימוד מתואר בפסבדו-קוד הבא:

שלב ה-Map

9.1 עבור כל הרשומות מיחס המרכזי של מכירות בצע מיפוי. המפתח יכיל מזהה של הרשומה מיחס המרכזי של מכירות, וכל מזהים של רשומות הנלוות.

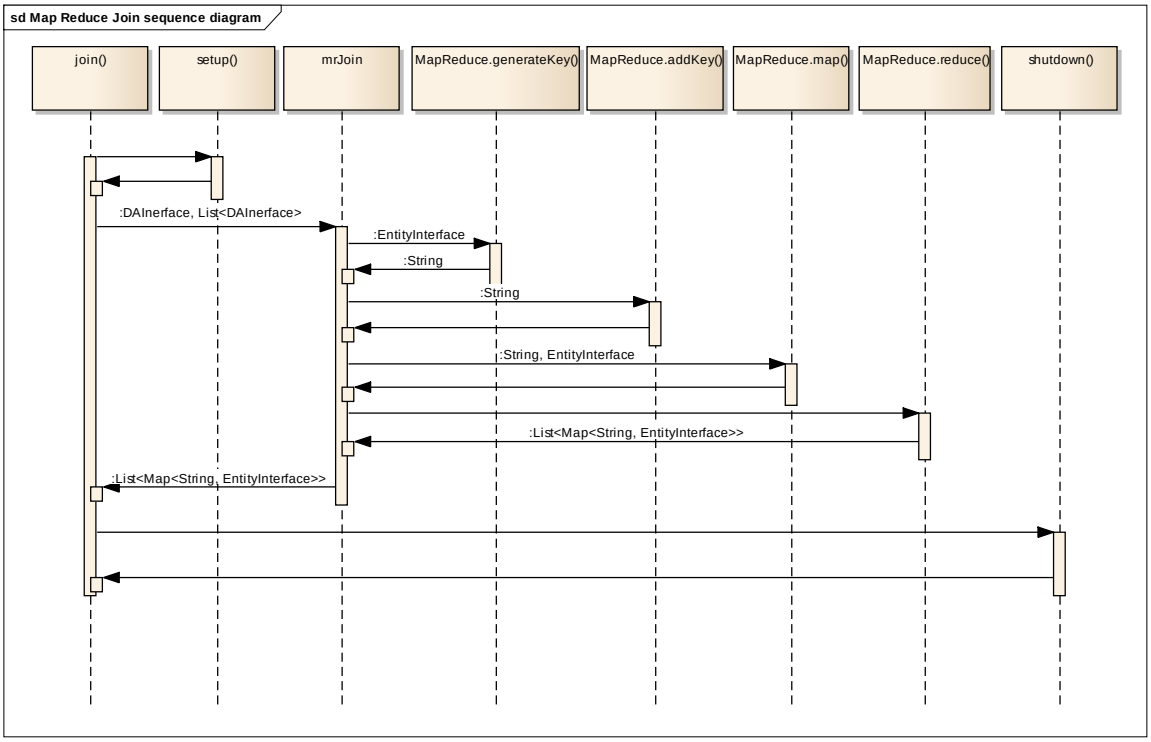
9.2 עבור כל הרשומות מיחסים אחרים של מוצרים, לקוחות, זמנים ומקומות בצע מיפוי.

שלב ה-Reduce

1. עבור כל הרשומה ממיפוי עבור היחס המרכזי חפס מיפוי מתאים לכל הרשומות הנלוות.

איור 12 מציג את תרשים זרימה לביצוע צימוד בבסיס נתונים של Berkeley לאחר מימוש של Map-Reduce.

איור 12 תרשים זרימה לביצוע צימוד בבסיס נתונים של Berkeley לאחר מימוש של Map-Reduce



4.8 השוואת ביצועים

לצורך השוואת ביצועים בין פעולת צימוד הקיימת ב-NoSQL ופעולת צימוד לאחר מימוש של Map-Reduce ב-NoSQL הרצנו שתי פעולות אל אותה כמות של נתונים. ההרצות התבצעו במיסגרת ריצה של תהליך אחד ותהליכון אחד. בפעולת צימוד לאחר מימוש של Map-Reduce ב-NoSQL כפי שנאמר קודם השתמשנו בתהליכון אחד, ולכן מספר התהליכי Reduce הוא אחד, כלומר k = 1, ועל פי סעיף 6.8:

a = 1
b = 1
c = 1
d = 1

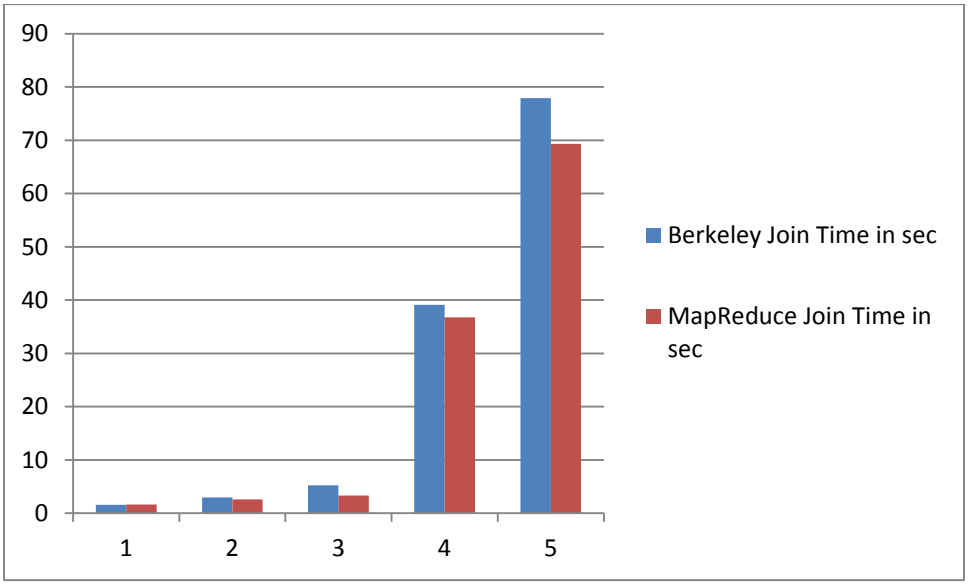
מכיוון שיש שימוש רק תהליך ה-Reduce אחד אין משמעות להפעיל את פונקציית ה- HASH ולכן לא השתמשנו באלגוריתם המקורי של אולמן

טבלה 1 ואיור 13 מסכמים את התוצאות.

טבלה 1 תוצאות של הרצות של שתי פעולות צימוד

מספר הרצה	מספר רשומות ביחס מרכזי	מספר רשומות ביחסים נלווים	זמן ביצוע של פעולת צימוד המקורית בשניות	זמן ביצוע של פעולת צימוד המשופרת בשניות
1	10,000	1,000	1.592	1.607
2	50,000	5,000	2.98	2.574
3	100,000	10,000	5.243	3.339
4	500,000	50,000	39.126	36.739
5	1,000,000	100,000	77.891	69.327

איור 13 תוצאות של הרצות בצורה גרפית



אפשר לראות שרק כאשר מספר רשומות ביחס המרכזי של מכירות היה 10,000 זמן הביצוע של פעולת הצימוד המקורי של היה יותר טוב.
בכל השאר של הרצות מימוש של Map-Reduce ב-NoSQL "ניצח", עם גידול של מספר הרשומות ההפרש בביצועים הולך וגודל לטובת ה-MapReduce.
<u>המסקנה</u>: מימוש של Map-Reduce בבסיס נתונים של NoSQL מייעל משמעותית את ביצועים של פעולת צימוד.

5. סיכום הפרויקט

בפרויקט זה הצגנו את עיקרון של NoSQL DB ו-Map-Reduce Framework. בנוסף גם הסברנו את מבנה של Berkeley DB ופרויקט עצמו.
מימשנו MapReduce ב-NoSQL DB. לפי תוצאות של הרצות אפשר לראות שהממשק (ה-API) החדש מייעל באופן משמעותי את פעולת צימוד (Join) בבסיס נתונים ה-NoSQL.

6. נספח א – אלגוריתם של ג'פרי אולמן [1]

מטרת המעמר של ג'פרי אולמן היא לחקור ולבחון את אלגוריתמים של פעולת צימוד עבור כמה יחסים בסביבת MapReduce, ובפרט המעמר עוסק עם אלגוריתמים שמקטינים.

6.1 מבוא

- **הגדרה של הסביבה**

להלן רשימת הגורמים המתארים את הסביבה בה תהליך של Map-Reduce יכול להתרחש:

1. **קבצים**.

קובץ הוא אוסף של רשומות. הדרישות עבור קבצים הם:

   a. גודל הקובץ יכול להיות מאוד גדול.
   b. סדר הרשומות בלתי צפוי.
   c. הרבה תהליכים יכולים לקרוא את אותו קובץ בו זמנית.

2. **תהליכים.**

תהליך הוא יחידת חישוב. תהליך יכול לקבל קלט מאחד או יותר קבצים, ולכתוב פלט לאחד או יותר קבצים.

3. **מעבדים.**

כל צומת במודל החישוב של Map-Reduce הוא צומת עם יחידת עיבוד מרכזית (CPU), זכרון ראשי (RAM) ואחסון משני (Secondary Storage). כל תהליך יכול להיות מוקצה לכל מעבד.

- **הגדרות עבור חישוב של זמן ריצה של האלגוריתם**

האלגוריתם מתבסס על מודל של הגמ"ל (גרף מכוון ללא מעגלים) של תהליכים כאשר ישנה קשת בין תהליך P_i ו- P_j אם"ם קלט של תהליך P_j הוא שווה או שווה חלקית לפלט של תהליך P_i (חלקית משמעות שתהליך P_j יכול גם לקבל קלטים נוספים מתהליכים אחרים).
התהליך לא יכול להתחיל עד שכל הקלטים שלו נוצרו.

- **עלות התקשורת של תהליך** (The communication cost) היא בעצם גודל הקלט לאותו תהליך.
  הערה: לא סופרים גודל הפלט של תהליך מכיוון שפלט של תהליך מסוים חייב להיות קלט לתהליך לכל הפחות אחד אחר (כך שגודל הזה סופרים כי עלות התקשורת), אלא אם כן מדובר על פלט סופי של האלגוריתם כאשר גודל הפלט סופי לא ניספר כלל כיוון שגודל הפלט סופי קשור לנתונים ולא לאלגוריתם עצמו.
- **עלות כוללת של התקשורת** (The total communication cost) היא סכום עלויות התקשורת של כל התהליכים המשתתפים באלגוריתם.
- **עלות מקסימלית של התקשורת** (The elapsed communication cost) מוגדרת לפי גרף מכוון. אם נסתכל על כל המסלולים ונחשב את עלות הכוללת של התקשורת עבור כל המסלולים אזי עלות כוללת של התקשורת המקסימלית היא העלות המקסימלית של התקשורת.

6.2 צימוד בין יחסים

- **צימוד ב-2 שלבים (2-Way Join) ב-Map-Reduce**

נתונים שני יחסים R(A, B) ו-S(B, C). על מנת לבצע את צימוד בין שני.
תהליכים של Map ממפים את כל רשומה (a, b) מ-R לתוך זוג עם מפתח b וערך (a, R).
באותו אופן תהליכים של Map ממפים את כל רשומה (b, c) מ-S לתוך זוג עם מפתח b וערך (c, S).
תפקיד של תהליכי Reduce הוא לצרף את רשומות מ-R ו-S כאשר יש להם ערך משותף של b. כלומר כל הרשומות עם אותו ערך של b נשלחות לאותו תהליך של Reduce. נניח שמשתמשים ב-k תהליכי של Reduce. אז נבחר את פונקציית גיבוב h הממפה ערכי B לתוך k תאים. כל ערך הגיבוב מתאים לתהליך של Reduce אחד. כל תהליך של Map שולח את תוצאות המיפוי עם ערך b לתהליך של Reduce לפי ערך הגיבוב h(b). תהליך של Reduce כותב את את תוצאה (a, b, c) לפלט.

- **צימוד בין יחסים בשלב אחד**

רוצים לבצע את צימוד של שלושה יחסים: R(A, B), S(B, C) ו-T(C, D).
אפשר לבצע צימוד על ידי סדרה של של שני 2-way joins. כלומר תחילה צימוד בין שני יחסים ראשונים, ואז לצמד את יחס שלישי עם התוצאה. לדוגמה תחילה צימוד בין R ו-S, ואז T עם התוצאה. הסדרה כזאת אפשר לבצע ב-Map-Reduce כפי שמוסבר בסעיף קודם.

ישנה דרך אחרת כאשר מבצעים את צימוד של שלושה יחסים בתהליך של Map-Reduce יחיד:
התהליך של Map שולח את כל רשומה מ-R ו-T לכל התהליכים של Reduce שונים, אבל כל רשומה של S נשלחת אך ורק לתהליך של Reduce אחד.
מצד אחד שכפול נתונים מעלה עלות התקשורת, אבל מצד שני אנחנו לא צריכים לשמור תוצאות של צימוד ראשון.
נניח שמשתמשים ב-k=m^2 תהליכים של Reduce עבור m איזה-שהוא.
ערכים של B ו-C יהיו מגובבים ל-m תאים, וכל תהליך של Reduce יהיה מחובר לזוג של תאים: אחד עבור B ואחד עבור C.
נניח h היא פונקציית גיבוב עם טווח 1, 2, …, m, וגם נחבר את כל תהליך של Reduce לזוג (i, j) כאשר i=1, 2, …, m ו-j=1, 2, …, m.
כל רשומה S(b, c) נשלחת לתהליך של Reduce שמספרו (h(b), h(c)).
כל רשומה R(a, b) נשלחת לכל התהליכים של Reduce שמספרם (h(b), x) עם x כלשהוא.
כל רשומה T(c, d) נשלחת לכל התהליכים של Reduce שמספרם (y, h(c)) עם y כלשהוא.
לכן כל תהליך (i, j) מקבל $\frac{1}{m^2}$ שורות של S, ו-$\frac{1}{m}$ שורות של R ו-T.
לדוגמה עבור m=4 באיור 14:

איור 14 צימוד בין שלושה יחסים עבור m = 4

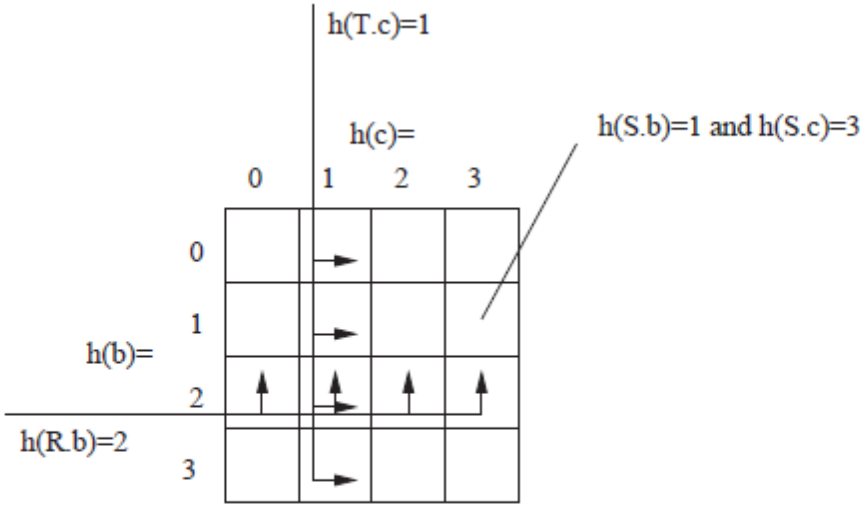


Figure 1: Distributing tuples of R, S, and T among $k = m^2$ processes

כל תהליך של Reduce מבצע את צימוד של רשומות שהוא קיבל. אם בין שלוש רשומות R(a, b), S(b, c) ו-T(c, d) מתבצעת פעלות צימוד, אזי הרשומות האלה נשלחות לתהליך של Reduce שמספרו (h(b), h(c)), ולכן הצימוד מחושב נכון.

- **אופטימיזציה של פעולת צימוד– דוגמה מקדימה**

בסעיף קודם בחרנו b ו-c על מנת ליצור את מיפוי, וגם בחרנו עבורם אותו מספר תאים $m = \sqrt{k}$. האם הבחירה זו הטובה ביותר ? כדי ללמוד איך לייעל מיפוי עבור פעולת צימוד נתחיל מדוגמה של צימוד מעגלי: $R(A, B) \bowtie S(B, C) \bowtie T(A, C)$

נניח שמספר המטרה של מפתחות מיפוי הוא k. כלומר, חייבים להשתמש ב-k תהליכי Reduce כדי לצמד את רשומות משלושה יחסים. כל אחת מהתכונות A, B ו-C תהיה חלק ממפתח. נסמן אותם כ-a עבור A, b עבור B ו-c- עבור C.
מניחים שקיימת פונקציית גיבוב שממפה את ערכים של התכונת A לתוך a תאים שונים, ערכים של התכונת B לתוך b תאים, וערכים של תכונת C בתוך c תאים.
<u>הערה</u>: **abc = k** .
כל תהליך של Reduce מחובר למפתח (u, v, w) כאשר u הוא ערך הגיבוב בטווח 1…a המייצג תאים לערכי תכונת A.
באותו אופן v הוא ערך הגיבוב בטווח 1…b המייצג תאים עבור B, ו-w הוא ערך הגיבוב בטווח 1…c המייצג את C. תהליך של Reduce כלשהוא יקבל את רשומת (x, y) מ-R רק כאשר h(x) = u ו-h(y) = v.
כלומר כל רשומה (x, y) מ-R תשלח ל-c תהליכים של Reduce המחוברים למפתח (h(x), h(y), w) כאשר w הוא בטווח 1…c.

באותו אופן כל רשומה (y, z) מ-S תשלח ל-a תהליכים של Reduce המחוברים למפתח (u, h(y), h(z)) כאשר u הוא בטווח a…1.
וכל רשומה (x, z) מ-T תשלח ל-b תהליכים של Reduce המחוברים למתח (h(x), v, h(z)) כאשר v הוא בטווח b…1.
מספר הרשומות הנשלחות מתהליכים של Map לתהליכים של Reduce הוא: rc + sa + tb
כאשר r הוא מספר רשומות ביחס R, s הוא מספר רשומות ביחס S, ו-t הוא מספר רשומות ביחס T.
צריכים להקטין את הביטוי rc + sa + tb כפוף לאילוץ abc = k.
<u>אילוץ נוסף הוא הבא</u>: כל מספרים a, b, c חייבים להיות חיובים.
נוכל להשתמש בכופלי לגראנז':

$rc + sa + tb - \lambda(abc - k)$

נפתור את הביטוי עבור שלושה משתנים a, b, c ונשווה ל-0. אז נקבל :

$s = \lambda bc$
$t = \lambda ac$
$r = \lambda ab$

אם נכפיל את ביטוים במשתנים חסרים אז נקבל:

$sa = \lambda k$
$tb = \lambda k$
$rc = \lambda k$

אם נכפיל את צד שמאל של שלושה ביטוים אז נקבל :

$rstk = \lambda^3 k^3 \Rightarrow \lambda = \sqrt[3]{rst/k^2}$

$sa = \lambda k \Rightarrow a = \sqrt[3]{krt/s^2}$

$tb = \lambda k \Rightarrow b = \sqrt[3]{krs/t^2}$

$rc = \lambda k \Rightarrow c = \sqrt[3]{kst/r^2}$

אז עלות התקשורת המינימלית היא:

$rc + sa + tb = 3\sqrt[3]{krst}.$

בנוסף אפשר לראות לדוגמה ש$\frac{a}{b} = \frac{t}{s}$. כלומר יחס בין משתנים הוא שווה יחס הפוך לגודל היחס ממנו התכונה חסרה.

- **השוואת ביצועים לסידרה של 2-Way Join**

על מנת לפשט את החישוב נניח שגודל של כל אחד משלושה יחסים שווה ל-r.
בנוסף נניח שהסתברות שלשתי רשומות מיחסים שונים ישנה תכונה משותפת היא p.
אזי עבור האלגוריתם המשופר עלות התקשורת היא:

6.2.1 3r עבור קלט לתהליכים של Map.

6.2.2 $3r\sqrt[3]{k}$ עבור קלט לתהליכים של Reduce.

בסה"כ עלות התקשורת עבור האלגוריתם המשופר היא $O(r\sqrt[3]{k})$.
עבור סידרה של 2-Way Join עלות התקשורת היא:

1. 2r עבור קלט לתהליכים של Map עבור 2 יחסים ראשונים, שזה גם קלט לתהליכים של Reduce עבור 2 יחסים ראשונים.

2. גודל הפלט של Reduce עבור 2 יחסים ראשונים הוא $r^2p + r$. לכן גודל הקלט תהליכים של Map עבור שלב שני יהי $r^2p + r$ כאשר r הוא עבור יחס שלישי. וזה בעצם קלט לתהליכים של Reduce עבור שלב שני (לא סופרים גודל של פלט סופי).
בהנחה ש-$rp > 1$ נקבל עלות התקשורת עבור הסידרה של 2-Way $O(r^2p)$.
אם נשווה ($r\sqrt[3]{k}$) מול r^2 נקבל את הגבלה $k < (rp)^3$. ההגבלה זאת אפשרית כיון ש-k הוא פרמטר שאנחנו יכולים לבחור, ו-r ו-p הם הפרמטרים השייכים לנתונים.
<u>דוגמה</u>: נניח $r = 10^7$, $p = 10^{-5}$, ו-$k = 1000$. אפשר לראות ש-$k < (rp)^3$.
עלות התקשורת עבור סידרה של 2-Way Join היא $r^2p = 10^9$, לעומת עלות התקשורת של אלגוריתם המשופר $r\sqrt[3]{k} = 10^8$.
גודל הפלט עבור דוגמה שלנו R(A, B) ⋈ S(B, C) ⋈ T(A, C) היא $r^3p^3 = 10^6$.

6.3 אופטימיזציה של Multiway Joins

- **אלגוריתם מקדים לאופטימיזציה של משתנים משותפים**

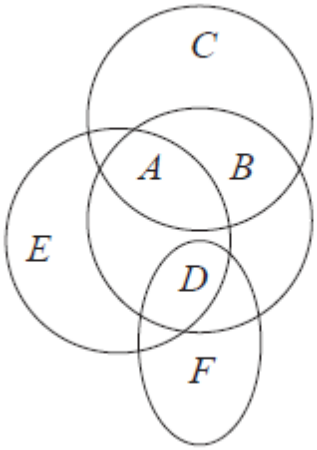
נניח רוצים לחשב את צימוד של היחסים $R_1, R_2, \ldots, R_n$, והתכונות בתוך היחסים הם A_1, A_2, A_n .
להלן הלגוריתם המקדים לאופטימיזציה של משתנים משותפים (נראה בהמשך את דוגמה של האלגוריתם המקדים) :

שלב 1:
התחל מביטוי עבור עלות התקשורת:
$\pi_1 + \pi_2 + \ldots + \pi_n - \lambda(a_1a_2\ldots a_m - k)$
כאשר π_i היא מכפלה של מספר רשומות r_i של יחס R_i במכפלה של משתנים משותפים a_j כך שיחס R_i לא מכיל תכונת A_j.

שלב 2:
פתור את הביטוי עבור כל משתנה משותף a_i, ותשווה את התוצאה ל-0. ואז תכפיל את המשוואה ב-a_i.
התוצאה היא הוסף של משוואות מצורת $S_{a_i} = \lambda a_1a_2\ldots a_m$ כאשר S_{a_i} הוא סכום של כאלה π_i-ים כך ש-A_i הוא לא חלק מ-R_i.
כיוון ש- $k = a_1a_2\ldots a_m$ נוכל לרשום $S_{a_i} = \lambda k$.
אלה בעצם המשוואות לגראנז' עבור פעולת הצימוד שלנו.

שלב 3:
כיוון ששלב 2 נותן לנו m משוואות עם m משתנים (a_i-ים) נוכל לפתור את המשוואות עבור a_i-ים במונחים של λ, k, וגודל היחסים r_i-ים.
נשתמש בדוגמה באיור 15 על מנת להראות את האלגוריתם:

איור 15 דוגמה לביצוע האלגוריתם המקדים



להלן יחסים המשתתפים בצימוד:
R(A, B, C), S(A, B, D), T(A, D, E), U(D, F)

נגדיר משתנים הבאים:
r – מספר רשומות ביחס R.
s - מספר רשומות ביחס S.
t - מספר רשומות ביחס T.
u - מספר רשומות ביחס U.
a, b, c, d, e, f – מספר תאים פונקציית גיבוב שממפה את ערכים של תכונות A, B, C, D, E, F בהתאם.

<u>שלב 1</u>: בונים את משוואה עבור עלות התקשורת:
rdef + scef + tbcf + uabce – λ(abcdef – k)

<u>שלב 2</u>: נפתור את המשוואות לגראנז':

עבור a:
ubce - λbcdef = 0 => ubce = λbcdef => aubce = λk

עבור b:
tcf + uace - λacdef = 0 => tcf + uace = λacdef => btcf + buace = λk

עבור c:
sef + tbf + uabe – λabdef = 0 => sef + tbf + uabe = λabdef => csef + ctbf + cuabe = λk

עבור d:
ref – λabcef = 0 => ref = λabcef => dref = λk

עבור e:
rdf + scf + uabc - λabcdf = 0 => rdf + scf + uabc = λabcdf => erdf + escf + euabc = λk

עבור f:
rde + sce + tbc - λabcde = 0 => rde + sce + tbc = λabcde => frde + fsce + ftbc = λk

שלב 3 של האלגוריתם שהצגנו אינו מושלם כיוון שפתרון עבור משתנים משותפים יכול להיות קטן מ-0.

בנוסף ישנם מקרים בהם למשוואות אין פתרון בר-ביצוע כיוון שכמה π_i-ים מתאפסים.

לדוגמה אם נפחית את משוואה רשאונה ממשוואה שנייה נקבל ש- *tbcf = 0*. אך זה לא יתכן כיוון שכל המשתנים משותפים וגדלים של יחסים הם מספרים חיובים.
על מנת לפתור את הבעיה נשתמש ברעיון של "תכונות נישלטות":

- **תכונות נישלטות**

תכונת *Y* נקראת נישלטת אם:

- כל יחס המכיל את תכונת *Y* גם מכיל את תכונת *X*.
- תכונת *Y* מופיעה אך ורק פעם אחת בפעולת צימוד.

בדוגמה שלנו תכונת *A* שולטת בתכונת *B*: תכונת *A* מופיעה ב-*R*, *S*, ו-*T*, אך תכונת *B* מופיעה ב- *R* ו-*S*. בנוסף לתכונת *B* גם תכונות *C*, *E* ו-*F* הן נישלטות: התכונות האלה מופיעות פעם אחת בפעולת צימוד.
עבור תכונה נישלטת אפשר להציב למשתנה משותף שלו 1.
<u>הערה</u>: משתנה משותף ששווה ל-1 לא יהיה חלק ממפתח במיפוי של פונקציית גיבוב כיוון שישנו רק תא אחד עבור תכונה זאת וערך זהה לכל המפתחות.
בסעיף הבא נוכיח שעלות התקשורת לא תגדל אם נשתמש בכלל של תכונה נישלטת..

- **כלל של תכונה נישלטת – הוכחה**

נניח שתכונה A שולטת בתכונה B.
נוכיח שאם נחליף את ה-a ב-ab ו-b ב-1 העלות לא תגדל.
ישנם 3 מקרים :

1. a ו-b הם גורמים בהמשוואות לגראננז'.
   הסיבה לכך היא שלא A ולא B נכללות ביחס.
   אזי לפני ההחלפה יש לנו מכפלה ab, וגם אחרי ההחלפה יש לנו אותה מכפלה:

ab = ab*1 = ab

2. רק b הוא גורם בהמשוואות לגראננז'.
   הסיבה לכך שרק A נכללת ביחס. בדוגמה שלנו זה tbcf.
   במקרה כזה אם נחליף b ב-1העלות תקטן.
3. לא a ולא b הם גורמים בהמשוואות לגראננז'.
   הסיבה לכך שגם A וגם B נכללות ביחס. בדוגמה שלנו זה rdef.
   במקרה כזה לא תהיה השפעה אם נשנה את a ו-b.

חשוב לציין שמקרה רביעי בו רק a הוא גורם לא יכול להתרחש כאשר A שולטת ב-B כיוון שאז B נכללת ביחס אבל A לא נכללת וזו סתירה להגדרת תכונה נישלטת.

- **האלגוריתם המלא**

שלב 1:
בחר תכונות שיהיו משותפות ויהיו חלק מפתח במיפוי. בשביל זה סלק תכונות נישלטות.

שלב 2:
התחל מביטוי עבור עלות התקשורת:
$\pi_1 + \pi_2 + \ldots + \pi_n - \lambda(a_1 a_2 \ldots a_m - k)$

כאשר π_i היא מכפלה של מספר רשומות r_i של יחס R_i במכפלה של משתנים משותפים a_j כך שיחס R_i לא מכיל תכונת A_j.

שלב 3:
בנה משוואות לגראננז'.

שלב 4:
פתור משוואות לגראננז'.

נמשיך אם הדוגמה שלנו:
R(A, B, C), S(A, B, D), T(A, D, E), U(D, F)

שלב 1:
מסלקים כל התכונות חוץ מ-A ו-D.

שלב 2:
עלות התקשורת היא:

rd + s + t + ua

שלב 3:
משוואות לגראננז' הן:

ua = λk
rd = λk

שלב 4:
כיוון ad = k נקבל:

ua*rd = uard = ruk
ruk = $\lambda^2 k^2$ => ru = $\lambda^2 k$

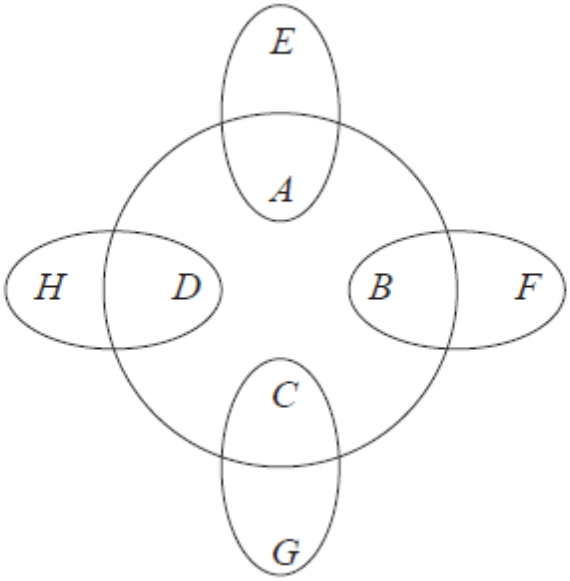
$\lambda = \sqrt{\frac{ru}{k}}$

a = $\sqrt{\frac{rk}{u}}$ **d =** $\sqrt{\frac{uk}{r}}$

6.4 צימוד בין יחסים מצורת כוכב (Star Join)

מקרה מיוחד הוא כאשר ישנו יחס מרכזי, ויחסים אחרים מחוברים אליו, כלומר צורת כוכב.
איור 16 מציג את דוגמה של צימוד הכוכב (Star Join).

איור 16 דוגמת צימוד הכוכב



אז עבור צימוד באיור 16 נקבל:

$R(A, B, C, D) \bowtie S(A, E) \bowtie T(B, F) \bowtie U(C, G) \bowtie V(D, H)$

עלות התקשורת היא:

$R + sbcd + tacd + uabd + vabc$

משוואות לגראנז' הן:

$tacd + uabd + vabc = \lambda k$
$sbcd + uabd + vabc = \lambda k$
$sbcd + tacd + vabc = \lambda k$
$sbcd + tacd + uabd = \lambda k$

אם נפחית כל ביטוי מכל ביטוי אחר נקבל:

$b = \frac{at}{s}$, $c = \frac{au}{s}$, $b = \frac{av}{s}$.

כיון ש-$abcd = k$ נקבל:

$a = \sqrt[4]{\frac{ks^3}{tuv}}$, $b = \sqrt[4]{\frac{kt^3}{suv}}$, $c = \sqrt[4]{\frac{ku^3}{stv}}$, $d = \sqrt[4]{\frac{kv^3}{stu}}$

6.5 סיכום

במאמר של ג'פרי אולמן חוקרים ובוחנים את אלגוריתמים של פעולת צימוד עבור כמה יחסים בסביבת Map-Reduce, ובפרט המאמר עוסק עם אלגוריתמים שמקטינים עלות כוללת של תקשורת.
ראינו איך אפשר לבצע את צימוד של כמה יחסים בו זמנית בצורה יעילה בעזרת כופלי לגראנז', כך שביצועים טובים יותר בהרבה לעומת Two-Way Join. בנוסף ראינו את אופטימיזציה של Multi-Way Joins בעזרת כלל של תכונות נשלטות, שהביא לעלות כוללת המינימלית של תקשורת.

רשימת מקורות

[1] Foto N. Afrati, Jeffrey D. Ullman: Optimizing joins in a map-reduce environment. Proceeding of EDBT 2010: 99-110

[2] http://metadata-standards.org/Document-library/Documents-by-number/WG2-N1501-N1550/WG2_N1537_SQL_Standard_and_NoSQL_Databases%202011-05.pdf
Presentation of Keith W. Hare, JCC Consulting, Inc.

[3] http://docs.oracle.com/cd/E17277_02/html/collections/tutorial/

[4**]** BerkeleyDB vs. FoundationDB
http://pages.foundationdb.com/berkeleydb_vs_foundationdb?gclid=CIqLotLS1cICFebLtAodkBkAKg