

אוניברסיטה הפתוחה
המחלקה למתמטיקה ומדעי המחשב

שיטה לאחסון נתוני NST המשודרים ממובייל לענן

עבודת מסכמת זו הוגשה כחלק מהדרישות לקבלת תואר
"מוסמך למדעים" M.Sc. במדעי המחשב
באוניברסיטה הפתוחה
החטיבה למדעי המחשב

מרץ 2016

לריסה קייקוב

304684418

העבודה הוכנה בהנחייתה של ד"ר מיה הרמן

תוכן עניינים

5.....	תקציר
6.....	1 מבוא
7.....	1.1 מובייל
7.....	1.1.1 מבוא
7.....	1.1.2 ארכיטקטורת אנדרואיד
9.....	1.1.3 סוגי ממשקים בטלפון חכם
10.....	1.1.4 חיישני מובייל
11.....	1.2 מחשוב ענן
11.....	1.2.1 מבוא
12.....	1.2.2 ארכיטקטורה
13.....	1.2.3 שירותי מחשוב
15.....	1.2.4 סוגי פריסה
16.....	2 מחשוב מובייל ענן
16.....	2.1 מבוא
18.....	2.2 ארכיטקטורה
19.....	2.3 סינכרון יישומים ניידים עם שירותי אחסון בענן
21.....	3 אחסון נתונים רפואיים ממובייל לענן
21.....	3.1 סוגי נתונים רפואיים
21.....	3.2 סוגיות הקשורות בנתונים רפואיים
21.....	3.2.1 היקף נתונים
22.....	3.2.2 פרטיות אבטחה וסודיות
23.....	3.3 בסיסי נתונים
23.....	3.3.1 NoSQL
24.....	3.3.2 השוואה בין NoSQL, NewSQL, RDBMS
26.....	3.3.3 מנועי NoSql מרכזיים
30.....	3.4 MapReduce
30.....	3.4.1 מבוא

30	3.4.2	מודל התכנות
36	3.4.3	MapReduce-ייתרונות וחסרונות
38	3.4.4	גירסאות ושיפורים
40	3.4.5	שכבות תוכנה הקיימות מעל MapReduce
42	3.5	Hadoop
42	3.5.1	מבוא
43	3.5.2	HDFS
47	3.5.3	Hadoop MapReduce
52	3.5.4	תזמון משימות
53	3.5.5	YARN
58	3.5.6	HBASE
59	3.5.7	השוואה בין Hadoop ל – RDBMS
60	4	סקר ספרות אודות יישומי מחשוב מובייל ענן
60	4.1	מבוא
60	4.2	מערכת לניהול תמונות רפאויות במובייל ענן
62	4.3	מערכת Wanda לניטור חולים בעלי אי ספיקה לבבית
67	4.4	מערכת לכריית נתוני א"קג בזמן אמת בענן
76	5	הצעה לשיפור מערכת TELE-NST
76	5.1	תאור מערכת TELE-NST
76	5.2	תאור ארכיטקטורת המערכת TELE-NST שפותחה בפרויקט מתקדם במדעי המחשב
78	5.3	נתוני מערכת TELE-NST
78	5.4	תאור הבעיות במערכת TELE-NST
78	5.5	הצעה לשיפור מערכת TELE-NST
84	6	סיכום, מסקנות והצעה להמשך מחקר
86	7	מקורות

רשימת האיורים

9	ארכיטקטורת שכבות באנדרואיד [7]	איור 1 :
13	ארכיטקטורת שכבות בענן [4]	איור 2 :
17	מכשיר נייד מחובר לשרת ענן מרוחק דרך האינטרנט [14]	איור 3 :
17	ענן משאבים וירטואלי הבנוי ממכשירים ניידים [14]	איור 4 :
18	Cloudnet המאפשר למכשירי מובייל לעקוף בעיות של השהייה ורוחב פס [14]	איור 5 :
19	ארכיטקטורת מובייל ענן [6]	איור 6 :
23	דוגמה לטבלאות קשרים מול בסיס נתונים NoSQL	איור 7 :
26	גוגל Trend לבסיסי נתונים NoSQL, NewSQL, RDBMS	איור 8 :
27	ארכיטקטורת Cassandra [37]	איור 9 :
31	שלבי זרימת מידע ב MapReduce [25]	איור 10 :
32	תשתית MapReduce [38]	איור 11 :
33	עיבוד מקבילי ב MapReduce [38]	איור 12 :
34	פונקציות מיפוי וצמצום המוגדרות על ידי המשתמש לספירת מילים [25]	איור 13 :
40	שאלת SQL ומקבילה שלה ב Pig Latin [19]	איור 14 :
40	שאלת HiveQL לספירת מילים [19]	איור 15 :
44	ארכיטקטורת HDFS [16]	איור 16 :
45	תרשים כתיבת קובץ ב HDFS [16]	איור 17 :
46	תרשים קריאת קובץ ב HDFS [16]	איור 18 :
47	MapReduce בגרסה 1.0 של Hadoop [13]	איור 19 :
49	שלבי זרימת מידע ב Hadoop MapReduce [16]	איור 20 :
50	ממשק פונקציית מיפוי [5]	איור 21 :
51	פורמט אינדקס של משימת מיפוי, וקובץ נתונים [5]	איור 22 :
54	ארכיטקטורת Hadoop בגרסאות 1.0 ו-2.0	איור 23 :
56	Yarn בגרסה 2.0 של Hadoop [31]	איור 24 :
58	מערכת אקו של Hadoop בגרסה 2.0 [13]	איור 25 :
61	ארכיטקטורת המערכת לניהול תמונות רפואיות [23]	איור 26 :
62	ארכיטקטורת מערכת Wanda [10]	איור 27 :
64	ממשק אנדרואיד במערכת Wanda [10]	איור 28 :
67	תרשים אק"ג עבור מחזור אחד של פעילות לבבית [28]	איור 29 :
69	אלגוריתם ליצירת אשכולות מתוך רצף של נתוני אקג לנבדק [28]	איור 30 :
71	זרימת המערכת בהיבט של MapReduce [28]	איור 31 :
73	תצוגה אינטואיטיבית של האשכולות [28]	איור 32 :
74	זמן ריצה ממוצע עבור מספר ליבות שונות [28]	איור 33 :
74	זמן ריצה ממוצע של שלבי אלגוריתם עבור מספר ליבות שונה [28]	איור 34 :
75	תאור ביצועי מערכת כפונקציה של גודל הבלוק ב HDFS [28]	איור 35 :
76	רכיבים עיקריים של מערכת TELE-NST [30]	איור 36 :
82	שידור נתוני NST במערכת Advanced TELE-NST	איור 37 :
83	ארכיטקטורה של מערכת Advanced TELE-NST	איור 38 :

ניטור צירים וקצב לב עוברי (NST- Nonstress Test) מבוצע במסגרת מרפאות ובתי חולים במהלך השבוע 25 עד שבוע 41 להריון. מטרת הניטור, לזהות מוקדם ככל האפשר נשים המצויות בסיכון מוגבר ללידה מוקדמת ו/או מצוקה עוברית. במסגרת פרויקט מתקדם של לימודים לתואר שני פיתחתי תת מערכת המטפלת בנתוני NST במסגרת מערכת TELE-NST. מערכת TELE-NST מאפשרת לבצע ניטור וניהול NST מרחוק באמצעות סלולר, העברת שידור מלא של המנויה לענן, שמירת נתוני השידור בבסיס נתונים רלציוני, וביצוע הליך של כריית מידע על נתוני השידור. אחת הבעיות העיקריות של המערכת שפותחה בפרויקט היתה הקושי להתמודד עם מידע רב. ככל שבסיס הנתונים גדל, חלה האטה בזמן עיבוד הנתונים עד שבסופו של דבר המערכת קרסה. בעיה נוספת, כריית מידע על נתוני NST לא מתבצעת בזמן אמת (העיבוד מתבצע על נתונים לאחר קבלת כל השידור, בעוד שקיים צורך במענה מיידי לאירועי צירים וקצב לב עוברי).

אחד הפיתרונות לטיפול בהיקף גדול של נתונים אותו יש לעבד במהירות וביעילות הוא המעבר לשיטת איחסון Hadoop MapReduce ולכן בעבודה זו נסקור את Hadoop ואת מודל התכנות MapReduce. בנוסף, בעבודה נסקרו טכנולוגיות מובייל (מעה"פ אנדרואיד), מחשוב ענן, ושילוב בין מובייל לענן בתחום הרפואי. כמו כן, נסקר נושא ה BigData, בהיבט של אופן התמודדות במהירות וביעילות כולל מתן פתרונות מגוונים של Big Data בענן. העבודה הנוכחית, מתמקדת בבסיס נתונים NoSQL, ובמנועים עיקריים לטיפול ב NoSQL. כמו כן, העבודה עוסקת בהבדל המהותי בין בסיס נתונים רלציוני ל NoSQL.

אחת ההמלצות העיקריות של עבודה מסכמת זו היא להמשיך בפיתוח מערכת TELE-NST ולבנות מחסן מידע בענן מעל Hadoop לאחסון נתוני NST. כמו כן, לבצע כריית מידע בזמן אמת במהירות וביעילות וזאת באמצעות אלגוריתם למיזוג ופיצול אשכולות דינמי, כאשר החישובים לבניית האשכולות, יבוצעו במקביל על מספר רב של צמתים באמצעות Hadoop MapReduce.

במסגרת פרויקט מתקדם של לימודים לתואר שני, פיתחתי מערכת TELE-NST המאפשרת לבצע ניטור וניהול נתוני צירים וקצב לב עוברי מרחוק באמצעות סלולר וזיהוי אוטומטי של נשים המצויות בסיכון מוגבר ללידה מוקדמת ומצוקה עוברית. נתוני ניטור צירים וקצב לב עוברי מורכבים מנתונים בדידים ונתונים רציפים, נסמן נתונים אלו כ-NST. נתוני NST נשמרים בענן בבסיס נתונים רלציוני, ועל הנתונים מבוצע הליך של כריית מידע. אחת הבעיות העיקריות של מערכת TELE-NST היתה הקושי להתמודד עם מידע רב. ככל שבסיס הנתונים גדל, חלה האטה בזמן עיבוד הנתונים עד שבסופו של דבר המערכת קרסה. בעיה נוספת, כריית מידע על נתוני NST לא מתבצעת בזמן אמת (העיבוד מתבצע על נתונים לאחר קבלת כל השידור, בעוד שקיים צורך במענה מיידי לאירועי צירים וקצב לב עוברי).

בעבודה זו, אסקור מהו Hadoop ואבחן את יישמותו לנושא איחסון נתוני NST. Hadoop היא פלטפורמה, המספקת תשתית לאחסון ועיבוד מקבילי על גבי אשכולי ענק של עד אלפי שרתים. באמצעות הכלים והטכנולוגיות שפלטפורמה זו מספקת, ניתן לקחת פעולה כבדה (מבחינת זמן וכמות הקלט/פלט הנדרש) ולפזרה על פני כמות גדולה של מחשבים חלשים במטרה לקבל במהירות את התוצאות הנדרשות. פלטפורמת Hadoop מתחלקת לשני פרויקטים עיקריים, ניהול מידע ע"י אחסונו במערכת קבצים מבוזרת בין שרתים פיזיים שמכונה HDFS (Hadoop File System) ומנגנון חישוב מקבילי Hadoop MapReduce. HDFS מהווה תשתית לפרוייקטים בעלי בסיס נתונים NoSQL, באמצעותו מחולקים קבצים גדולים לקבצים קטנים יותר, המפוזרים בין השרתים השונים עבור קריאה מקבילית של המידע בין השרתים. NoSQL מהווה תחליף לבסיס נתונים רלציוני בפתרון Hadoop, תומך בביצועים טובים עם ארכיטקטורה מבוזרת הניתנת להרחבה, פיתרון פשוט להבנה ולתחזוקה, ופיתרון טוב במקרים של ניתוחים סטטיסטיים. Hadoop MapReduce מספק תשתית מבוזרת לעיבוד מקבילי. התשתית אחראית על עיבוד נתונים וניהול משאבי אשכול. התשתית מחלקת משימות של הקלט על ידי שימוש ב HDFS ומפזרת את הפעילות על כלל השרתים. תשתית זו מסוגלת להתמודד עם שגיאות בזמן ריצה כגון נפילת שרתים או בעיות רשת. בגרסאות חדשות של Hadoop נוספה שכבת Yarn המהווה מסגרת לניהול תהליכים ומשאבי אשכול במקום ה Hadoop MapReduce. הוספת Yarn מהווה פיתרון נוח למגבלות של Hadoop, ומאפשר ניצול טוב יותר של אשכול.

בעבודה מסכמת זו, נחקרו נושאים המצויים בחזית המחקר והמסייעים לשיפור מערכת TELE-NST שפותחה במסגרת פרויקט מתקדם של לימודים לתואר שני.

בעבודה ידונו הנושאים הבאים :

בפרק 1 נסקרה טכנולוגיית מובייל ומחשוב ענן.

בפרק 2 נסקר שילוב בין מובייל לענן.

בפרק 3 נסקרו בסיסי נתונים ושיטות לאחסון נתונים רפואיים במובייל ענן .

בפרק 4 נסקרו מערכות קיימות לאחסון נתונים רפואיים במובייל ענן.

בפרק 5 ניתנה הצעה לשיפור מערכת TELE-NST שפותחה במסגרת פרויקט מתקדם במדעי המחשב.

1.1 מובייל

1.1.1 מבוא

מובייל הינו כל סוג של מכשיר נייד כגון טלפון חכם, לפטופ אלחוטי ועוד. שילוב בין טכנולוגיית מובייל לבין הגישה לאינטרנט הוביל לפופולריות של המכשירים. הסיבה לשימוש הגובר של מחשוב הנייד היא היכולת שלו לספק כלי למשתמש בכל רגע נתון ללא קשר למקום הימצאו. עם זאת, מחשוב נייד סובל מבעיות רבות כגון זמינות מוגבלת, ההגבלה במשאבים (חיי סוללה, איחסון, רוחב פס), בתקשורת (ניידות, אבטחה, ניתוקים תכופים), ובכח עיבוד נתונים חלש של המכשירים הניידים. במגזר הבריאותי, ההשפעה של מגבלות אלו חשובה בשל הגודל המסיבי של הנתונים, מורכבות הנתונים, וקצב גידול מואץ בכמות הנתונים. כתוצאה מכך מגוון רחב של יישומים רפואיים כבדים לריצה על מכשירים ניידים [2], [1]. מה שדרוש, היא סביבת ענן המסוגלת לאחסן, לחפש, לשתף, ולנתח נתונים בקנה מידה גדול ביעילות.

1.1.2 ארכיטקטורת אנדרואיד

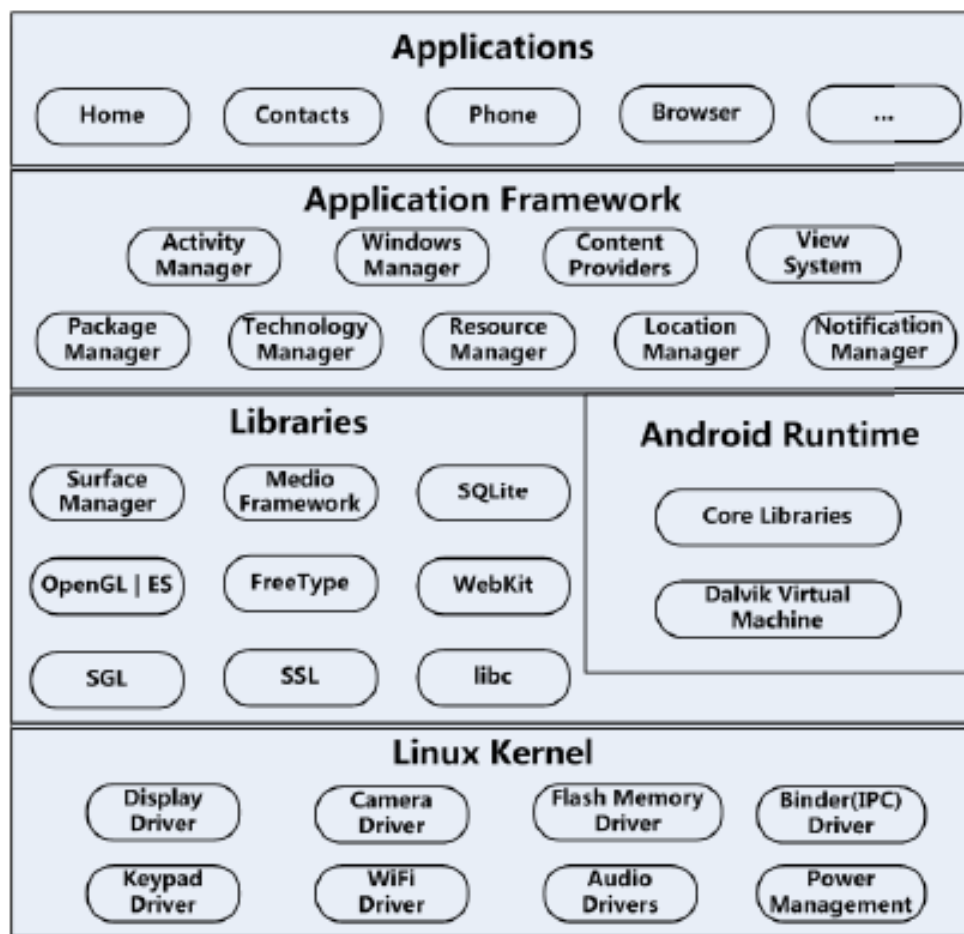
מעה"פ אנדרואיד הינה קוד פתוח ופותחה על ידי גוגל. אנדרואיד רצה על גרעין לינוקס, יישומי אנדרואיד כתובים בשפת JAVA. גוגל עיצבה מחדש והתאימה את ה - DVM (Dalvik virtual machine, בדומה ל jvm, מכונה וירטואלית המפרשת ומריצה java bytecode) למאפייני החומרה של המכשירים הניידים. באיור 1 ניתן לראות את ארבעת השכבות המרכיבות את מערכת הפעלה של אנדרואיד:

- שיכבה עליונה היא **שכבת היישום**, שכבה זו מכילה את היישומים של המערכת כגון מייל, SMS, מפות, דפדפן, אנשי קשר ואחרים. כל היישומים כתובים בשפת Java.
- מתחתיה נמצאת **שכבת תשתית היישום** המכילה:
 - סט של Views, המשמשת לבניית יישומים עם ממשק משתמש עשיר
 - סט ספקי תוכן, המאפשרים ליישומים לגשת לנתונים מיישומים אחרים
 - מנהל משאבים, המספק גישה למשאבים לא מקודדים
 - מנהל התראות, המאפשר ליישומים להציג התראות בשורת המצב
 - מנהל הפעלה, המנהל את מחזור החיים של היישומיםשכבה זו, מנהלת את הפונקציות הבסיסיות של הטלפון כגון ניהול שיחה ומשאבים.
- מתחת לשכבת תשתית היישום נמצאת **שכבת הספריות ו - Android Runtime**.

שכבת הספריות מכילה ספריות הכתובות בשפת ++c. קומפוננטות שונות באנדרואיד משתמשים בספריות, הספריות משמשות גם כתמיכה לשכבת תשתית היישום .
Android Runtime מכילה ספריות גרעין המערכת ואת Dalvik virtual machine. כל הקלאסים של Java בזמן ריצה מועברים לפורמט dex, המשלב את הקבצים ומוריד קוד מיותר. באנדרואיד יכולים להיות כמה jvm, כאשר כל אחד מהם רץ בתהליך לינוקס נפרד (יישום אנדרואיד רץ במופע של מכונה וירטואלית). ספריות הגרעין, מאפשרות למפתחים לכתוב יישומי אנדרואיד באמצעות שפת תכנות java סטנדרטית.

- בשיכבה התחתונה **גרעין מעה"פ** הממומש בלינוקס. הגרעין משמש כשכבת הפשטה בין החומרה לתוכנה, ומכיל את הפונקציונליות של המערכת הכוללת ניהול זיכרון, אבטחה, ניהול תהליכים וכ"ד [7].

משתמש מתקשר עם שכבת היישום, עבור פונקציות בסיסיות כגון ביצוע שיחת טלפון, גישה לדפדפן האינטרנט, שאר השכבות נגישות בעיקר עבור מפתחים. היישום מתחבר ישירות עם שכבת התשתית, המנהלת את הפונקציות הבסיסיות של הטלפון כגון ניהול שיחה ומשאבים. שכבת תשתית היישום מתקשרת עם שכבת הספריות, המכילות סט של הוראות המנחות את המכשיר בטיפול בסוגים שונים של נתונים כגון השמעה והקלטה של תכני וידאו (מטופל על ידי ספריית מדיה). הסיפריות מתקשרות עם שיכבת הגרעין המקשר בין התוכנה לחומרה ומספק את הפונקציונליות העיקרית של המערכת.



איור 1 – ארכיטקטורת שכבות באנדרואיד [7]

1.1.3 סוגי ממשקים בטלפון חכם

בכל טלפון חכם קיימים שלושה סוגי ממשקים USB, Bluetooth, ו-Wi-Fi. USB (Universal Serial Buses) ממשק סטנדרטי לחיבור חוטי, Bluetooth ו-Wi-Fi ממשקים סטנדרטיים לחיבור אל חוטי.

בחיבור חוטי, יש צורך בחיבור פיזי, באלחוטי, המידע מועבר דרך מדיום סביבתי ללא צורך בחוטים. **USB** - תקן חיבור בין מחשבים להתקני ציוד היקפי. שיטה מהירה ביותר, אמינה ולא צורכת סוללה (ה USB מטעין את הסוללה)

Bluetooth – תקן תקשורת אל חוטית, ליצירת רשת במרחב האישי, ברמת אבטחה גבוהה על ידי שימוש בגלי רדיו. שימושי בהעברת מידע בין מכשירים קרובים, כאשר אין בעיית מהירות כגון טלפונים, מדפסות מודמים. טכנולוגיית bluetooth זולה יותר מ-Wi-Fi וצורכת פחות סוללה.

Wi-Fi - טכנולוגיה המאפשרת למכשירים אלקטרוניים להעביר נתונים באופן אלחוטי באמצעות גלי רדיו. Wi-Fi יכולה לעבוד עם כמה מכשירים במקביל, פחות מהירה וצורכת הרבה סוללה. היתרון המשמעותי על Bluetooth ו-USB הוא מרחק התקשורת [9].

1.1.4 חיישני מובייל

הטלפונים החכמים מכילים סוגים שונים של חיישנים, הפופולריים ביניהם הם מד תאוצה, ג'ירוסקופ, מגנטומטר, מיקרופון ומצלמה [11].

מד תאוצה - מכשיר המודד תאוצה, רגיש לכוחות הגרויטציה שמופעלים עליו ויודע לתרגם זאת לערך ווקטורי. מד תאוצה מאפשר לטלפון חכם לדעת את מצבו במרחב, אם הוא מוחזק כלפי מעלה, או על הצד דוגמא כאשר מביטים בתמונה.

ג'ירוסקופ - מכשיר למדידת אוריינטציה, מאפשר למדוד ולזהות תנועות סיבוב ולאתר את מיקומו ותנועותיו של אלמנט בתנועה.

מגנטומטר - משמש למדידת הכח וכיוון של שדות מגנטיים, מאתרים את מיקומם ביחס לצפון המגנטי של כדור הארץ. החשיבות של החיישן בתיקון טעויות זיהוי מיקום של הג'ירוסקופ.

מיקרופון - חיישן הממיר קול לאות חשמלי

מצלמה - לוכדת תמונה על ידי זיהוי תבנית, ומעבדת את התבנית על ידי אלגוריתמים שונים.

בעזרת שלושת החיישנים, תאוצה, ג'ירוסקופ ומגנטומטר, הטלפון החכם יכול להעריך כל סוג של תנועה המתבצעת בו. דוגמאות ליישומים מבוססי שלושת חיישנים: חיפוש אנשים בציבור (Trace track), מדידת תאוצת המהירות בזמן נסיעה (Dangerous drive), זיהוי לחיצת אצבע (Phone Hack).

1.2.1 מבוא

במחשוב, לעיתים קרובות דרוש מספר עצום של משאבים לביצוע משימות בקנה מידה גדול או בכדי לצמצם סיבוכיות לזמן סביר.

בתחילה, בעיות אלו טופלו בעזרת מחשבים בעלי ביצועים גבוהים. בהמשך, פותחה טכנולוגיית **רשת מחשוב** (Grid Computing) הבנויה על רשת מכונות היכול להימצא במרחקים גיאוגרפיים גדולים.

ברשת מחשוב קיים שיתוף משימות בין מספר גדול של מחשבים המקושרים ברשת תקשורת. משימה יכולה להיות כל דבר החל מאחסון נתונים ועד לעיבודים מורכבים שניתן לפזרם בין מחשבים שונים.

רשת מחשוב הציגה יכולות חדשות כגון שימוש דינאמי בשרתים, היכולת להסתמך על מספר רב של משאבים השייכים למתחמים שונים, ומציאת הסט הטוב ביותר של מכונות העונות על הדרישות של היישומים.

לרשת מחשוב יכולה להיות הגבלה במערכות הפעלה ובשירותים המוצעים. למרות שרשת מחשוב משתמשת דינאמית בשירותים, יש לה סט מוגבל של אפשרויות זמינות, ולפעמים הם לא גמישים מספיק בכדי לכסות את כל הצרכים.

לעומת זאת, הרעיון העיקרי **במחשוב ענן** הוא שימוש במשאבים בהתאם לצרכים. המשאבים נמצאים במקום מרוחק והגישה עליהם נעשת דרך רשת מחשוב ולא במחשב מקומי.

מטרת הענן, לאפשר למשתמשים להתחבר לכל שירותי המחשוב להם זקוקים באמצעות האינטרנט. המשתמשים לא צריכים לרכוש ולנהל משאבים ומערכות מחשוב, במקום זאת הם שוכרים אותם כשירות מספקים.

לענן קיימת יכולת דינאמית לשנות את כמות המשאבים לפי דרישה. יישומים דורשים רמה בסיסית של משאבים, בתנאי עומס יש צורך במשאבים נוספים. ללא שימוש בענן, צריך לבנות יכולת המספיקה להתבצע כראוי תחת עומסים כבדים עם ביצועים טובים, בענן ניתן להתחיל עם כמות קטנה של משאבים ולגדול בהתאם לצרכי היישום.

לענן קיימת יכולת אוטומטית (על ידי שימוש ב API) לספק, לפרוס ולשחרר מופע של מכונה וירטואלית. יישום בענן, הוא בעל יכולת לבקש מופעים וירטואלים חדשים לפי הצורך והמשאבים האלה מובאים בצורה מקוונת תוך דקות. לאחר שהביקוש דועך ואין צורך במשאבים, המופעים הוירטואלים נלקחים ואין צורך יותר לשלם עליהם.

בענן התשלום הוא לפי צריכה, אין חוזה שנתי ואין התחייבות לרמה מסוימת של צריכה. ניתן להקצות משאבים לפי צורך ולשלם עבורם על בסיס שעות. יזמים מתחילים לא צריכים הון התחלתי, יכולים לנצל כמויות אדירות של משאבים ולשלם עליהם גרושים לפי שעה.

נבחין בין סוגי פריסה שונים של ענן: **ענן פרטי**, תשתית השייכת לארגון יחיד, **ענן ציבורי**, הענן בבעלות ספק וזמין לציבור הרחב בתשלום או בחינם **וענן היברידי**, שילוב של שניים או יותר סוגי פריסה. המגמה כיום היא שימוש בענן היברידי. על פי סקרים [35], 55 אחוז מהארגונים מתכננים לעבור לענן היברידי.

בהיבט הרפואי, מחשוב ענן מציע יתרונות משמעותיים: בתי חולים ומרפאות דורשות גישה מהירה למחשוב, וכמות איחסון ללא הגבלה שאיננו מסופק בהגדרות מסורתיות. יתר על כן, נתונים בריאותיים צריכים להיות משותפים על פני אזורים גאוגרפיים שונים בכדי לחסוך עיכוב משמעותי בטיפול בלקוחות. מחשוב ענן עונה על כל הדרישות, המאפשרים לארגוני בריאות לשפר שירותים ללקוחות וחולים, על ידי שיתוף מידע ויעילות תפעול. החסרונות העיקריים בענן רפואי הם איומי אבטחה, פרטיות וסודיות. לדוגמא, דליפת מידע אודות תגובה לתרופות של חברה אי' לחברות תרופות אחרות, מסחר בנתוני חולים, סיכון לאובדן נתוני חולים ועוד. חסרונות נוספים הם זמינות נתונים ותלות באינטרנט מהיר. כיום ספקי ענן משקיעים הרבה מאוד משאבים באבטחה, סודיות ופרטיות, לדוגמא בגוגל רק לחלק קטן מהעובדים קיימת גישה למידע, והם עוברים פוליגרף מדי שבוע.

1.2.2 ארכיטקטורה

באיור 2 מוצגות ארבעה שכבות שונות המרכיבות את מחשוב הענן [4]. השכבה התחתונה **שכבת המערכת**, מאופיינת על ידי משאבים פיזיים עליהם התשתית בנויה. המשאבים יכולים להיות מרכזי נתונים, שרתים החוברים יחדיו (clusters) ושרתים חילופיים. שכבה זו היא "כח המחשוב" של הענן [4]. מעליה נמצאת **שכבת הגרעין**, בה מנוהלת התשתית הפיזית. מטרת שכבה הגרעין, לספק סביבת עבודה ליישומים ולנצל את המשאבים הפיזיים בדרך הטובה ביותר. שכבת הגרעין מתבססת על טכנולוגיית וירטואליזציה. וירטואליזציה היא הפעלת תוכנה באופן בלתי תלוי בתשתיות המחשב, באמצעות דימוי של תשתיות אמיתיות. וירטואליזציה נותנת יכולת לבנות מחשבים לוגיים שבהם סביבת הרצה לא יודעת שהיא רצה בתוך אלמנט לוגי ולא מחשב אמיתי. וירטואליזציה ברמת החומרה מאפשרת בידוד יישומים וחלוקה טובה של משאבים פיזיים כגון CPU וזיכרון באמצעות מכונות וירטואליות. וירטואליזציה ברמת התוכנה מנהלת ביצוע של יישומים שפותחו מטכנולוגיות ספציפיות או שפות תכנות. שכבה גרעין בנויה משרותים הבאים: איכות השירות, בקרת כניסה, ניהול ובקרה של ביצועים, חשבנאות וחיוב. שכבת הגרעין ביחד עם שכבת מערכת מספקים פלטפורמה עליה יישומים פרוסים בענן. מעל שכבת הגרעין נמצאת **שכבת הביניים המשתמש**, המספקת סביבה וכלים המפשטים את הפיתוח והפריסה של יישומים בענן, לדוגמא ממשקים של Web2, ספריות, שפות תכנות. מעליה נמצאת **שכבת המשתמש** המכילה את היישומים בענן. כל שיכבה מתקשרת רק עם השכבה שנמצאת מתחתיה. יישום בענן, דרך שיכבת הביניים של המשתמש (לדוגמא ממשקי web 2.0), ניגש לשירותים של שיכבת הגרעין (הלוגיקה העיקרית של המערכת). השירותים של שכבת הגרעין משתמשים ומתקשרים עם התשתית הפיזית כגון המשאבים השונים.



איור 2 - ארכיטקטורת שכבות בענן [4]

1.2.3 שירותי מחשוב

ניתן לסווג את שירותי הענן לשלושה סוגים עיקריים [12], [4]:

תשתית כשירות (IAAS) - הספקת תשתית IT, מבוססים על משאבים וירטואלים או פיזיים, כסחורה ללקוח. המשאבים יכולים להיות שרת, טכנולוגית תקשורת, ציוד תקשורת, זיכרון, יחידת עיבוד, אחסון או שרת נתונים. המשאבים מנוהלים במרכז הנתונים של הספק. המשתמשים מחויבים על פי רמת השימוש ומגדירים את המערכות שלהם מעל משאבים אלו. התשתית מאפשרת שינוי גודל דינאמי של המשאבים לצרכי הלקוח בנקודות זמן שונות. מכונות וירטואליות הם הצורה הנפוצה ביותר של מתן משאבים למשתמשים, אותם הם יכולים באופן מלא להגדיר ולנהל לפי הצרכים שלהם.

פלטפורמה כשירות (PAAS) - פלטפורמת פיתוח בה משתמש יכול ליצור את יישומיו אשר ירוצו על הענן.

PAAS מספקת תשתית אפליקטיבית וסט של APIs הנותנת למפתחים יכולת לתכנת יישומים עבור הענן.

משתמש שרוצה להעלות את התוכנה שלו לענן, צריך לממש את ה-APIs שהענן מספק. כל ספק ענן מספק APIs משלו, יישומים שפותחו עבור ספק ענן ספציפי לא ניתן לנייד לענן אחר. אולם כיום מפותחת מגמה בענן המאפשרת להעביר יישום כיחידה אחת שלמה בין העננים. הפרוייקט בגוגל נקרא Google Container Engine, המשתמש ב-Kubernetes, מערכת לניהול יישומי קונטיינר ברחבי מחשבים מארחים, ו-Dockers, פרוייקט פתוח המבצע פריסה של היישומים בתוך קונטיינר.

ישנם מקרים בהם PAAS משמשת כמערכת משולבת המציעה שכבת פיתוח וגם תשתית IT עליה ירוצו התוכנות. על ידי העלאת תוכנה לענן, המשתמשים נהנים משירותים שענן מספק כגון שינוי גודל אוטומטי, איזון עומסים, שירותי אימות, שירותי תקשורת או קומפוננטות לממשק משתמש הגרפי. אחד הספקים המרכזיים המספקים פלטפורמת שירות הוא Google Apps Engine. Google Apps Engine היא פלטפורמה לפיתוח יישומי אינטרנט הניתנים להרחבה, הממומשת בסביבת הענן של Google. Google מספק סט של APIs ומודל אפליקציה המאפשרים למשתמש לקבל שירותים נוספים כגון דואר. המשתמשים יכולים לפתח אפליקציות בשפות תכנות, Python, Java ו-Jrubi [12], [4].

תוכנה כשירות (SAAS) - תוכנה כשירות נמצאת בקצה העליון של מחשוב ענן, ומספקת למשתמשי הקצה שירות משולב הכולל חומרה, פיתוח פלטפורמות ויישומים. למשתמשים אין אפשרות להתאים אישית את השירות, משתמשים יכולים רק לקבל גישה ליישום ספציפי. מפעילים את התוכנה מאתר הספק דרך רשת תקשורת. השימוש בתוכנה כשירות מקטין עלויות משום שלא נדרשת התקנת שרתים במרכז המחשבים של הארגון המשתמש, ובכך נחסכת ההתקנה והתחזוקה של מוצר תוכנה והתשלומים עבור התוכנה העשויים להיות זולים יותר.

ממשק המשתמש לשירותי SAAS ברוב המקרים Web יחד עם זאת אפליקציות רבות פועלות בשיטת Remote Desktop שבו ממשק המשתמש הוא חלוני והאפליקציה עצמה מאורחת אצל ספק השירות, האפליקציה לא "רצה" בצורה מקומית על מחשב הלקוח אלא בשרתי ספק השירות. דוגמא לשימוש ב-SAAS הם השירותים המסופקים על ידי Google עבור אופיס, כגון המסמכים של Google, קלנדר של Google, המסופקים חנם למשתמשי אינטרנט [4].

לדוגמא אם רוצים לעבור לענן הציבורי של גוגל:

עבור IAAS - תשתית והבסיס יועברו לענן (כמו לקחת מחשבים פיזית ולשים בענן). גוגל מספקת חומרה, לא אחראית על שידרוגים ועידכון גרסאות.

עבור PAAS - מעבירים קצת יותר, הענן אחראי גם על שידרוגים ועדכון גרסאות. התשתית חושפת API למימוש, דוגמא Google Apps Engine פלטפורמה לפיתוח יישומי אינטרנט בענן.

עבור SAAS - מדובר על צריכת שרות מלאה, לקוח לא משאיר שום אחריות אצלו, הכל עובר לענן. דוגמא google apps (gmail, docs...)

במוסדות רפואיים, הבחירה בשירות ענן תלויה בגודל הארגון. עבור אירגונים קטנים, SAAS היא האפשרות הכלכלית האטרקטיבית ביותר, לאירגון כדאי לשלם פר שרות מאשר לתחזק אנשי IT במשרה מלאה.

לאירגונים גדולים יותר, PAAS היא האפשרות העדיפה. לאירגונים יש את המשאבים הדרושים, לפתח פיתרון מבוסס ענן משלהם. במוסדות רפואיים המחפשים רק תשתית רחבה יותר, IAAS מציעה פיתרון חיסכוני, המספק יכולות הרחבה, גמישות, הסכם מוגדר ברמת השירות, גיבוי והגנה על נתונים [24], [8].

בכל שירותי המחשוב קיימים איומי אבטחה, פרטיות וסודיות, יחד עם זאת לכל אחד קיימת בעיית אבטחה ייחודית [42]:

תוכנה כשירות – ניהול סיסמאות. היישומים נמצאים בענן, קיימים מספר רב של סיסמאות גישה ליישומים (מערכת פגיעה לשימושים לא מורשים).

פלטפורמה כשירות – הצפנת מידע. כדי להשתמש בשירות קיים הצורך להצפין את כל המידע המועבר (כולל רשומות רפואיות, מספרי ביטוח לאומי וכ"ד). הצפנת נתונים פוגעת בביצועי המערכת. תשתית כשירות – השתלטות עוינת של שירותים. תשתית כשירות מתמקד בניהול מכוונות וירטואליות, העובדים עלולים לגשת למידע שאינם רשאים להשתמש בו.

1.2.4 סוגי פריסה

ענן פרטי - ענן פרטי היא תשתית השייכת לארגון יחיד. השימוש העיקרי בענן פרטי מיועד לארגונים המעוניינים לשמור על המידע שלהם בלי שהמידע יהיה חשוף לפרצות אבטחה שיכולות להיות קיימות בשירותי ענן ציבורי. השירות ניתן למספר מוגבל של משתמשים והמידע נגיש רק לעובדי הארגון או לעובדים מורשים מטעמן ואינו מאוחסן במקום שבו יהיה קל לפורצים מחוץ לארגון להגיע אליו. הפונקציונאליות בענן לא חשופה באופן ישיר למשתמש, אם כי בכמה מקרים שירותים עם תכונות משופרות בענן יכולים להיות חשופים, לדוגמה EBAY. ענן פרטי אידיאלי עבור מוסדות רפואיים לשמירה על המידע הסודי שלהם, החיסרון, התשתית בענן יקרה יותר לעומת שאר סוגי פריסה [12].

ענן ציבורי - הענן בבעלות ספק וזמין לציבור הרחב בתשלום או בחינם. בענן ציבורי, הספק מספק שירות, כגון יישום או מקום אחסון אשר יהיה זמין מכל מקום דרך האינטרנט. במוסדות רפואיים בעייתית להשתמש בענן ציבורי בגלל איומי אבטחה היות והענן פתוח לכלל הציבור. דוגמאות לענן ציבורי: Amazon, Google Apps, Windows Azure [12].

ענן קהילתי - ענן המשתף מידע בין מספר ארגונים בעלי תחומי עיסוק משותפים. עננים קטנים יכולים להרוויח מענן קהילתי בו גופים שונים תורמים את התשתית שלהם. עננים קהילתיים יכולים לצבור עננים ציבוריים או לספק תשתיות של משאבים. ארגונים קטנים יכולים לבוא ביחד כדי לאגר את המשאבים שלהם ולבנות ענן קהילתי פרטי, בניגוד, משווקים יכולים לאגר משאבי ענן מספקים שונים ולמכור אותם. תשתית הענן משותפת לכמה ארגונים, זה עוזר להפחית עוד יותר את עלויות בהשוואה לענן פרטי.

לדוגמא משרדי ממשלה שונים, הדורשים גישה לנתונים זהים של אוכלוסייה מקומית כלשהי או מידע הקשור לתשתיות, כגון בתי חולים, יכולים לנצל את ענן קהילתי ולנהל יישומים ונתונים בענן [12].

ענן היברידי - שילוב של שניים או יותר סוגי עננים (פרטי, קהילתי או ציבורי) כאשר כל אחד מהם הוא ישות נפרדת, אך השילוב שלהם מאפשר ליהנות מההטבות שכל סוג מציע בנפרד. בניגוד לענן ציבורי המאפשר לארגונים להעביר חלק מתשתיות שלהם לספקים בענן (הארגונים מאבדים שליטה על משאבים, נתונים וניהול קוד), עננים היברידיים המורכבים מתשתית המעורבת של ענן פרטי וציבורי, המאפשרת לארגונים לשמור על שליטה של נתונים רגשיים ולצמצם בצורה מקסימלית העברת תשתיות לענן. ענן היברידי מאפשר ליישומים ונתונים לעבור בקלות מענן אחד למשנו. במוסדות רפואיים לדוגמא, ניתן לשמור מידע קליני קנייני בענן פרטי ולהתשמש בענן הציבורי, כדי לחלוק את המידע עם החולים, רופאים וכ"ד [12].

המגמה כיום היא ענן היברידי, לפי סקרים אחרונים, 55 אחוז מהארגונים מתכננים לעבור לענן היברידי, 13 אחוז מתכננים להשתמש בענן ציבורי, 14 אחוז מתכננים לעבור לענן פרטי ולכל השאר אין תוכניות לעבור לענן [35].

2 מחשוב מובייל ענן

2.1 מבוא

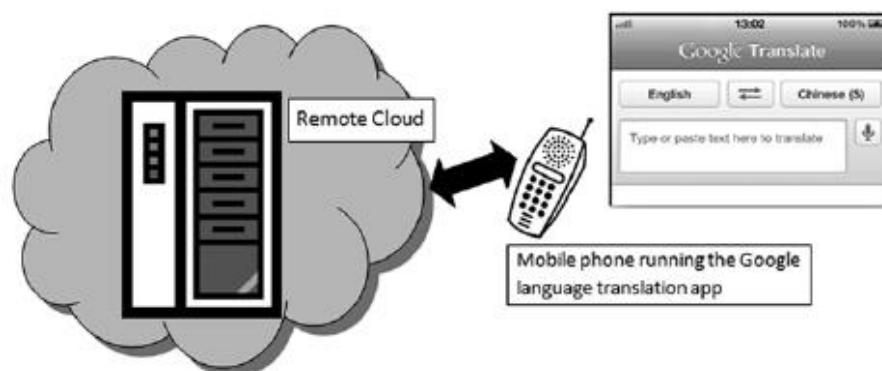
מחשוב מובייל בענן היא טכנולוגיה המשלבת את היתרונות של מובייל וענן, בה תשתית כח המחשוב והאיחסון מועברים ממכשיר מובייל לענן. במובייל ניתן לגשת ליישומים אלו דרך חיבור אלחוטי או דפדפן אינטרנט. ישנן מספר הגדרות למחשוב מובייל ענן. בדרך כלל מחשוב מובייל ענן פירושו להריץ יישום כגון gmail במובייל עם משאבים הנמצאים בשרת מרוחק, כאשר המובייל הוא לקוח המתחבר לשרת מרוחק בענן על ידי הדור השלישי (פרוטוקולי תקשורת לקשר רדיו), כמודגם באיור 3 [14]. גישה זו, נפוצה ביישומים רפואיים, המשתמשים בטלפון הנייד כאמצעי להעברת מידע לענן, ומעקב אחר נתונים בכל רגע נתון ומכל מקום. גישה אחרת, להסתכל על מכשירים ניידים בתור משאב של ענן היוצרים רשת עמית לעמית המודגמת באיור 4 [14], בה כל אחד מהקצוות מתפקד הן כלקוח והן כשרת (המכשירים הניידים בונים את הענן, באמצעות המשאבים שלהם). לדוגמא ענן הבנוי משרת וקבוצה של מכשירים ניידים. כל מכשיר הוא צומת, המחובר ברשת אלחוטית מקומית ומתפקד כמשאב של ענן. כאשר מגיעה משימה לביצוע, השרת מחלק את המשימה בין הצמתים השונים (ע"פ תכונות אופייניות של כל צומת, מהירות מעבד, מצב סוללה וכ"ו).

Cloudnet היא גישה נוספת, המודגמת באיור 5 [14]. Cloudnet, ענן מקומי קטן הבנוי מ plug computers (שרתים קטנים פחות חזקים וזולים יותר) המקושרים לשירותי ענן מרוחקים. המובייל מתחבר ל cloudnet ומעביר לו את כל עומס העבודה שלו במקום להתחבר לענן מרוחק. Cloudnet יכולים להיות מותקנים בתשתיות ציבוריות כגון בתי קפה, כך שהמכשירים ניידים יוכלו להתחבר, ולתפקד בתור לקוח ל Cloudnet, במקום להתחבר לענן מרוחק עם בעיות שהייה ורוחב פס [14].

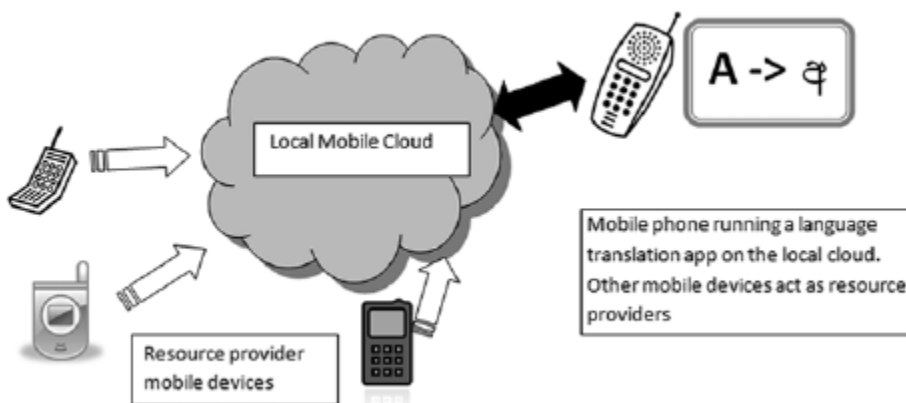
מחשוב מובייל בענן בתחום הרפואי, מעודד את השימוש בשירותי רפואה מרוחק ומפחית את הבעיות השכיחות ביישומים רפואיים כגון איחסון פייז קטן, מהירות חישוב, אבטחה ופרטיות [24].

מובייל בענן מאפשר לבתי חולים וארגונים רפואיים שונים, גישה לשרותים בענן מכל מקום ולא בשרתים המקומיים שלהם [2]. דוגמא למערכת רפואית המיושמת במובייל ענן הינה, שירותי ניטור רפואיים המחוברים לטלפון הנייד על ידי תקשורת אלחוטית ומאפשרים לחולים להיות במעקב בכל רגע נתון ומכל מקום [6].

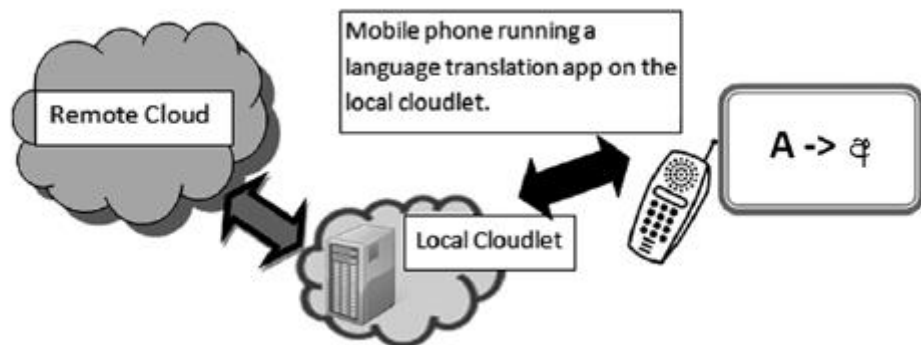
החסרונות העיקריים בשימוש של אחסון נתונים במובייל ענן הם בעיות אבטחה ופרטיות במיוחד בנתונים רגישים כדוגמת נתונים רפואיים, בקרת זרימת מידע, עלויות בלתי צפויות ותלות בספק [2].



איור 3 – מכשיר נייד המחובר לשרת ענן מרוחק דרך האינטרנט [14]



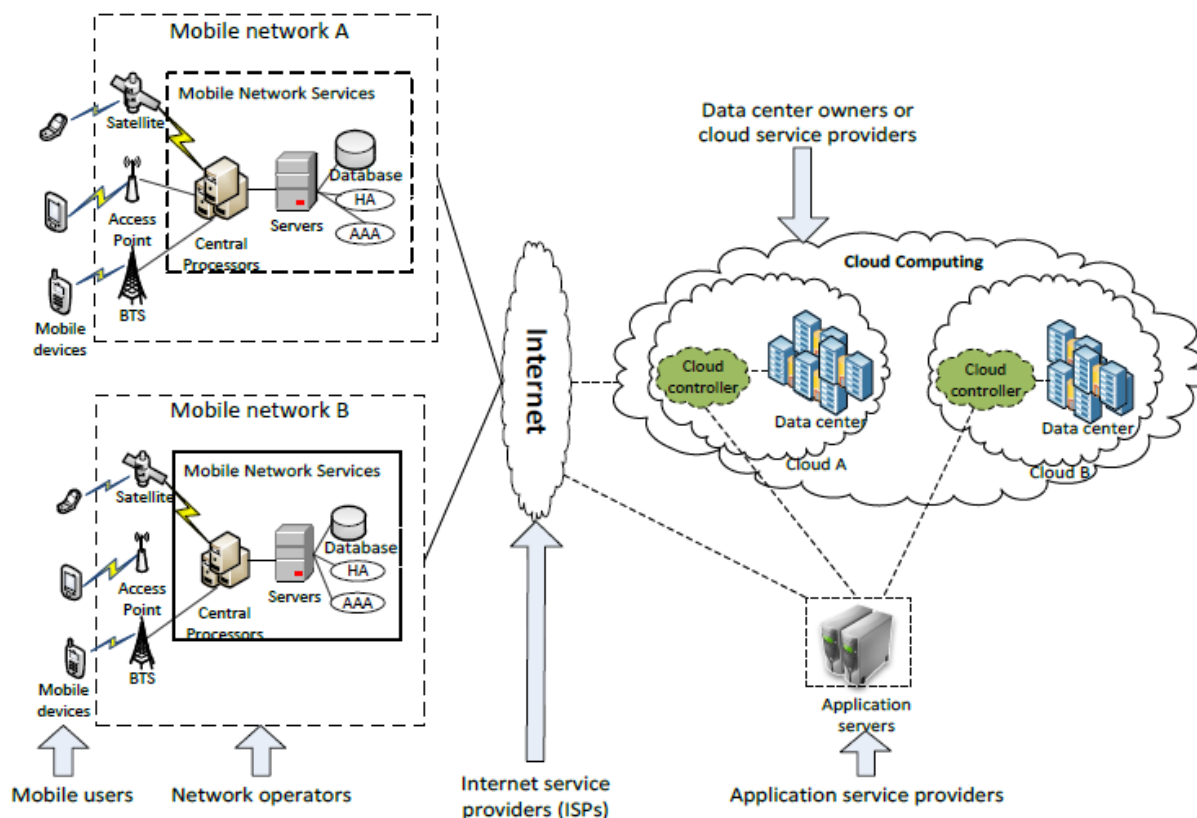
איור 4 – ענן משאבים וירטואלי הבנוי ממכשירים ניידים [14]



איור 5 – Cloudnet המאפשר למכשירי מובייל עקיפת בעיות השהייה ורוחב פס [14]

2.2 ארכיטקטורה

באיור 6 מודגמת ארכיטקטורת מובייל ענן [6]. מכשירי מובייל מחוברים לרשתות סוללריות באמצעות תחנות בסיס (access point, bts) היוצרים ושולטים בחיבורים (חיבורי אוויר) בין הרשת למכשירים. בקשות ומידע של משתמשים מועברים למעבדים מרכזיים (central processors) המחוברים לשרתים (Servers) המספקים שירותים של רשתות סוללריות. מפעילי רשתות סוללריות מספקים שירותים למשתמשי מובייל בתור AAA (אימות, אישור, וחשבונאות) מבוססי ספק ביתי (HA) והנתונים נשמרים בבסיסי נתונים של הרשת הסוללרית. לאחר מכן, בקשות מנויים מועברים לענן דרך האינטרנט. בענן, בקרי ענן (cloud controller) המעבדים את הבקשה ומפעילים את שירות המיחשוב המתאים (SAAS, IAAS, PAAS).



איור 6 – ארכיטקטורת מובייל ענן [6]

2.3 סינכרון יישומים ניידים עם שירותי אחסון בענן

המכשירים הניידים מאפשרים גישה לענן בעזרת הרשת דרכה הם מחוברים. שירותי הענן ניתנים על ידי ספקי שירות שונים. הבחירה של ספק ענן לאיחסון נתונים מהמובייל תלויה בצידוד שהספק מספק, מרחב הזיכרון, וניסיון של המשתמש בספק הענן. בבחירת שירות איחסון בענן, צריך להתחשב בקריטריונים הבאים:

- התאמת הפלטפורמה עבור שירות הענן המתאימה להתקן הנייד. לדוגמה iCloud מאפשרת רק ללקוחות אפל לאחסן בו נתונים בענן.
- סוגי מידע הניתנים לשמירה בענן, לדוגמה Amazon Cloud drive מאפשרת לשמור רק תמונות או קבצי מוסיקה
- פונקציונליות בענן שתאפשר מעקב ושיחזור גירסאות, סינכרון תיקיות מקומיות ושיתוף פעולה עם משתמשים אחרים.

- פונקציונליות של אבטחה שתאפשר למשתמשים להגן על הקבצים באמצעות סיסמאות, הצפנה ושיתוף קבצים באופן פרטי.

מעה"פ IOS של אפל משתמשת ב iCloud לשירותי הענן, מעה"פ Windows Phone של Microsoft משתמשת בפיתרון הענן skyDrive, מעה"פ של אנדרואיד משתמשת בענן של גוגל, מעה"פ של BlackBerry בשירותי הענן של BlackBerry Cloud. הערכה הגוברת בעולם היא שכאשר איחסון הנתונים בענן יהיה זול יותר, המחשבים הפרטיים ייעלמו ויוחלפו בצג או בטאבלט נייד, ללא חומרה לאחסון מקומי, והגישה לנתונים תהיה דרך הענן בלבד(כבר היום גוגל מוכרת מוצר כזה בשם Chromebook). המשתמשים יוכלו להתחבר מכל מקום בעולם למידע הפרטי שלהם, בתנאי שקיים חיבור זמין לרשת. בפיתרון הנ"ל קיימת בעיית אבטחה שעדיין צריך להתגבר עליה. במקרה והמכשיר הנייד נעלם, הפרטיות של המשתמש יכולה להיפגע. לכן בהתקנים ניידים יצרני חומרה ותוכנה יישמו פתרונות כדי להתגבר על הבעיה. לדוגמא איתור מכשיר אבוד בעזרת הרשת והצגתו על המפה, סיסמא למכשיר, גיבוי נתונים בענן, שליחת הודעה למכשיר לנעילת המכשיר/מחיקת המידע [15].

3 אחסון נתונים רפואיים ממובייל לענן

3.1 סוגי נתונים רפואיים

ברפואה קיים מגוון רחב של סוגי נתונים: נתונים בדידים, רציפים ותמונות. כיום ברוב המקומות הנתונים הרפואיים נשמרים בשרת מרכזי של מוסד רפואי [22].

- נתונים בדידים
 - נתוני טקסואליים - נתוני מידע שנאספו עבור טיפול בחולים, לדוגמא היסטוריה של מחלות, פרטיים אישיים של מטופלים.
 - נתונים נומריים - פרמטרים כגון בדיקות מעבדה, סימנים חיוניים (כגון טמפרטורה ודופק), מדידות שנלקחו במהלך בדיקה גופנית.
- נתונים רציפיים – נתונים אנלוגיים כדוגמת א.ק.ג (ECG) וכ"ו. בנתונים אלה נישמר גם מידע אודות תדירות האות, משרעת ואמפליטודת האות ועוד.
- תמונות – רנטגן, אולטרא סאונד, MRI, PET-CT וכ"ו. התמונות מורכבות מפיקסלים.

3.2 סוגיות הקשורות בנתונים רפואיים

3.2.1 היקף נתונים

ב 2012 נתוני בריאות דיגיטליים בעולם הוערכו בהיקף של 500 פטה בתים. מעריכים שבשנת 2020 היקף הנתונים יגדל ל 25,000 פטה בתים [1]. מאז ימיו הראשונים של פרויקט הגנום האנושי, ניהול נתונים הוכר כאתגר מרכזי למחקר בביוולוגיה מולקולרית. בעשור האחרון קיימת עלייה דרמטית בנתונים רציפים בגלל התקדמות הטכנולוגיה, ריבוי מכשירים, נגישות המידע, פלטפורמות ענן המאפשרות גישה ל Big Data, שיפור בביצועי כריית מידע ועוד.

מערכת Big Data היא לרוב מערכת אנליטית או מערכות המכילות מידע גולמי רב, אותו יש לעבד במהירות וביעילות. מערכת Big Data מתחלקת ל 3 ממדים עיקריים המכונים שלושת ה-V-ים [18]:

- נפח המידע - מערכות בעלות כמות גדולה מאוד של נתונים (פטה-בייטים).

- מהירות היווצרות מידע חדש - בעולם בו כל שנייה קיימים עשרות ציורים פייסבוק היכולת להתמודד עם שטף המידע היא אתגר גדול.
- כמות וסוגי מידע שונים הנאספים.

האתגר הגדול ב Big Data הוא אופן האיחסון והאחזור של כמויות אדירות של מידע. האתגר המשני הוא אופן עיבוד המידע, טעינה מהירה וחיפוש בתוכו פריט מסוים בזמן סביר כולל ניתוח המידע שיהווה מידע שימושי.

3.2.2 פרטיות אבטחה וסודיות

בעת שמירת נתונים בענן, קיימת אפשרות שהספק ימסור מידע לגורמים חיצוניים ללא ידיעתו של המשתמש. החשש הגדול במחשוב ענן הוא אבטחה ופרטיות. משתמשים רבים חוששים למסור את הנתונים שלהם לגורם שלישי. הדאגה גדולה יותר כאשר חברות מעוניינות לשמור על המידע הרגיש שלהן בענן. למרות שספקי השירות מבטיחים כי השרתים שלהם שמורים מורוסים ותוכנות זדוניות עדיין קיים חשש לאור העובדה כי קיימת גישה לשרת למספר משתמשים ברחבי העולם. פרטיות הוא נושא נוסף בענן, ובו דנים כיצד ניתן להבטיח שלנתוני לקוח יגשו רק משתמשים מורשים. לשם כך, ספקי שירותי ענן פיתחו חשבונות מוגנים על ידי סיסמאות ושרתי אבטחה, דרכם מועברים נתונים ושיטות הצפנה.

בענן רפואי מתעצם האתגר של איומי אבטחה כגון דליפת נתוני חולים ושימוש במידע בלתי מורשה. נתונים רפואיים בענן בעיקר נתוני חולים צריכים להיות מוגנים. נסקור מנגנוני אבטחה עיקריים בענן רפואי [1]:

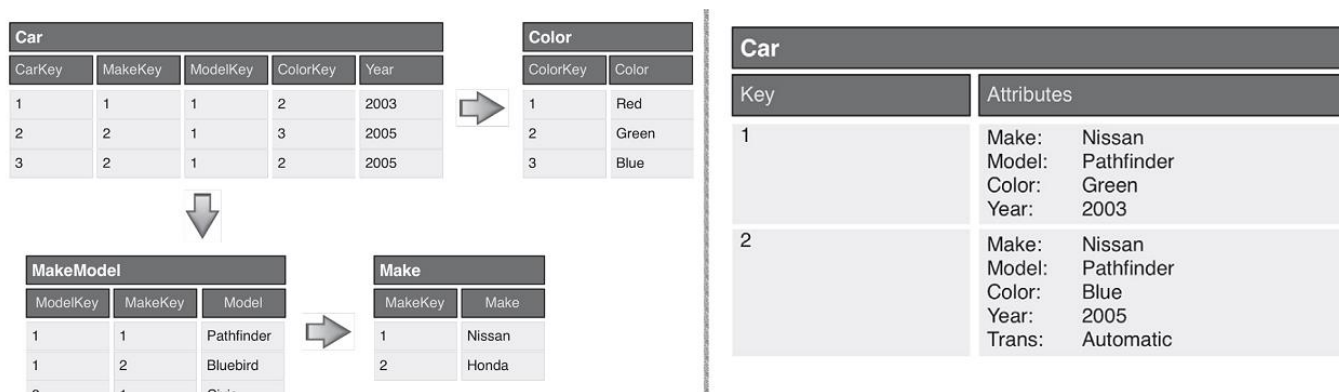
- אימות – כניסה לספקי שירות בענן וזיהוי משתמשים צריכים להיות מאומתים בכל כניסה על ידי משתמש וסיסמא.
- הרשאות – הרשאות כניסה ברמת משתמשים
- אי התכחות – אף צד לא יוכל להתכחש לשליחת/קבלת מידע. השיטה בדרך כלל מיושמת על ידי חתימה דיגיטלית, חותמות זמן, הצפנה וקבלת אישור.
- שלמות וסודיות- שלמות, שמירה על דיוק ועיקביות בנתונים רפואיים. סודיות, נגישות למידע רק למשתמשים מורשים. שלמות וסודיות מושגים על ידי בקרת כניסה והצפנה.
- זמינות – מידע צריך להיות זמין בעת הצורך. מערכות עם זמינות גבוהה שואפות להישאר זמינות בכל העת, למרות מכשולים רבים כגון הפסקות חשמל, כשלים בחומרה, שידרוג מערכות.

כיום ספקי ענן דואגים לאבטחה מאוד גבוהה, בגוגל לדוגמא רק לחלק קטן מהעובדים קיימת גישה למידע, והם עוברים פוליגרף מדי שבוע, בנוסף גוגל מאפשר העברת מידע מוצפן לענן, כך שמפתח ההצפנה נשאר בידי הלקוח.

3.3.1 NoSQL

NoSQL הוא בסיס נתונים לא רלציוני ובדרך כלל מבוזר. הרעיון המרכזי של NoSQL היא היכולת לשמור ולאחזר מידע במהירות רבה, וזאת על ידי חלוקת המידע למקורות אחסון. המידע מפוזר לרוחב על ידי שימוש במספר רב של שרתים שאינם תלויים אחד בשני [21].

נתוני NoSQL נשמרים במגוון רחב של פורמטים. החשובים שבהם איחסון לפי מסמכים, גרפים, ועל ידי צמידים של מפתח וערך. בגישה של מפתח, ערך יוצרים בסיס נתונים ללא קשרים כך שכל הנתונים הרלוונטיים לפריט מסוים מאוחסנים תחת אותו מפתח. ניתן לראות את ההבדלים (איור 7 צד שמאל) בין בסיס נתונים רלציוני בעל קשרים בין הטבלאות, לבין בסיס נתונים NoSQL (איור 7 צד ימין).



איור 7 – דוגמא לטבלאות קשרים מול בסיס נתונים NoSQL

ב NoSQL עבור כל פריט (מכונת) מרכזים את הנתונים שלו תחת אותו מפתח. התכונות מועתקות בין הפריטים, גישה זו משפרת יכולת הרחבה על ידי ביטול הצורך לצרף בין טבלאות שונות. ניתן לחלק את הנתונים בין שרתים שונים, לשכפל נתונים בלי להרוס את שלמות הקשרים בין הנתונים.

כדי לתחקר מערכות NoSQL לעיתים יש צורך להביא כל הערכים לפי מפתח. ל NoSQL יש יכולת מוגבלת לסנן את התוצאות לפי המידע שבערכים, שזה גם יתרון, אין צורך באופטימיזציה של שאילתות, סכמות ואינדקסים. ב Hadoop משתמשים ב MapReduce המאפשרת לנתח את המידע מחוץ ל NoSQL בצורה מהירה על כמה שרתים במקביל ובכך לייעל את השימוש במידע הרבה יותר מהר מ RDBMS (בסיסי נתונים רלציוניים).

לעיתים קרובות בסיסי נתונים מסוג NoSQL מממשים רק שלושה מתוך ארבעת הפרמטרים של ACID (עמידות, בידוד, עקביות, אוטומיות). היכולת לוותר על כל אחד מהפרמטרים מאפשרת לבסיסי נתונים להיות הרבה יותר מהירים, לבזר את המידע למספר שרתים ולהוות תחליף ראוי במקרים מסויימים.

3.3.2 השוואה בין NoSQL, NewSQL, RDBMS

RDBMS הוא בסיס נתונים רלציוני הבנוי מטבלאות, כאשר כל טבלה מכילה מידע על ישות מסוימת. RDBMS המסורתי יכול להבטיח ביצועים בסדר גודל של אלפי טרנזקציות בשנייה אבל לא מסוגל להתמודד עם עיבוד טרנזקציות מקוונות (OLTP) הכוללות קרוב למיליוני טרנזקציות בשנייה. לדוגמא, פרסום בזמן אמת, גילוי הונאות, משחקים מרובי משתתפים, ניתוח סיכונים [34].

RDBMS תומך ב ACID (תכונות של בסיס נתונים, עמידות, בידוד, עקביות, אוטומיות), ומבטיח שלמות נתונים בכל מחיר. בסיסי נתונים NoSQL שנסקרו בסעיף 3.3.1 מוותרים על אחד מארבעת הפרמטרים של ACID (לדוגמא עקביות) לטובת גמישות באיחסון ומפחיתים את המגבלות הקשות שיש ל RDBMS כגון איחסון במבנה טבלאי עם שורות והגדרת נתונים קפדנית. בסיסי נתונים NoSQL תומכים בביצועים טובים עם ארכיטקטורה מבוזרת הניתנת להרחבה. גם ב RDBMS קיימים בסיסי נתונים התומכים בארכיטקטורה מבוזרת הניתנת להרחבה כגון Oracle Rac. Oracle Rac הוא פתרון המשלב זמינות גבוהה (כאשר שרת נופל, האחרים מגבים אותו) ויכולת הרחבה (יכולת להתמודד עם יותר משתמשים ועם עומס רב יותר ע"י הוספת עוד שרתים למערך שכולם אקטיביים). הפיתרון של oracle rac מאוד יקר, וקיימת לו מורכבות רבה בהתקנה ותחזוקה.

NewSQL הוא בסיס נתונים חדש התומך ב SQL ו ACID וביצועיו ברמה של NoSQL. קיימים בסיסי נתונים NewSQL כדוגמת Nuodb, אך עדיין אין הוא עובד. בטבלה 1 מודגמים מאפיינים שלושת המערכות שנסקרו באופן השוואתי.

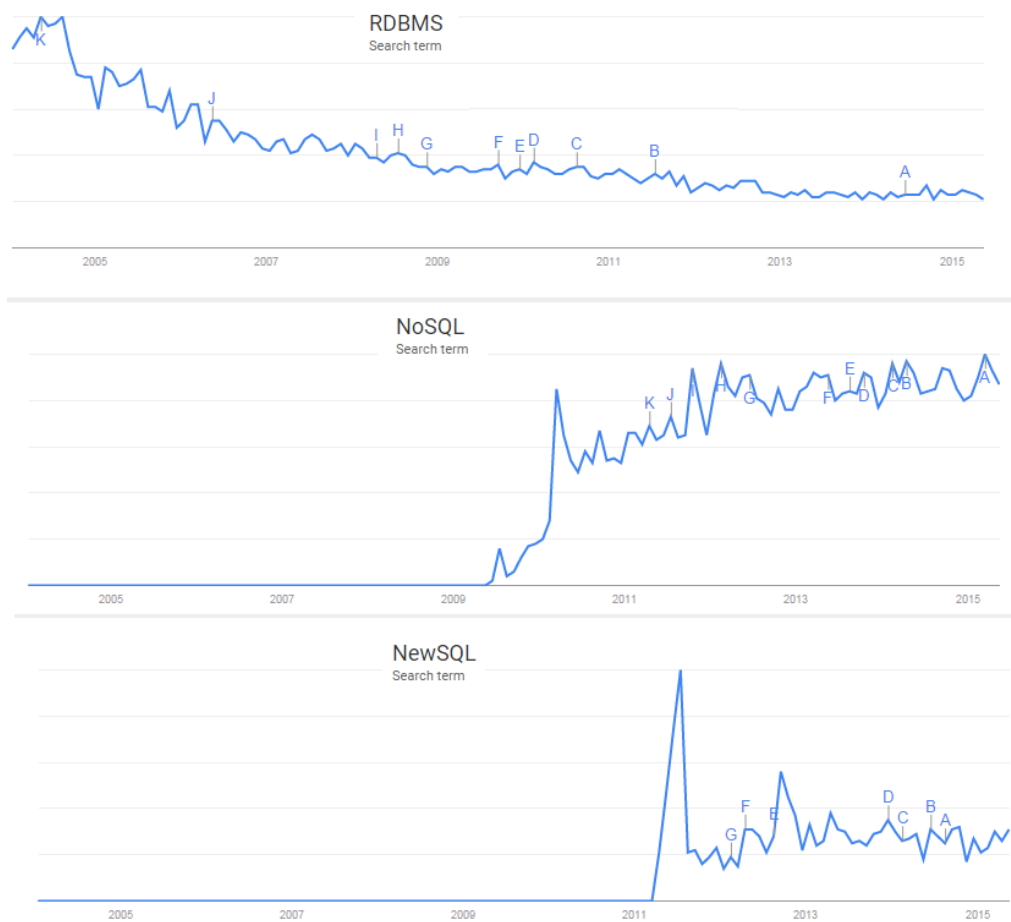
מאפיינים	RDBMS	NoSQL	NewSQL
תמיכה ב ACID	כן	לא	כן
OLTP (עיבוד טרנזקציות מקוון)	כן	לא	כן
אנליזה לנתונים (צבירה וכד)	כן	לא	כן
סכימה קשיחה (מודל מיפוי קפדני)	כן	לא	אולי
פורמט נתונים גמיש	לא	כן	אולי
מחשוב מבוזר	כן	כן	כן
יכולת הרחבה	כן	כן	כן
ביצועים בגידול מהיר של נתונים	מהיר	מהיר	מאוד מהיר
תקורה של ביצועים	גדולה	מתונה	מנימלית
פופולריות	גדולה	גדלה עם הזמן	גדלה מאוד לאט

טבלה 1 : ניתוח השוואתי בין NoSQL, RDBMS, NewSQL [36]

קריטריונים לבחירת סוג בסיס הנתונים לשימוש

- הקפדה על שלמות נתונים ועיקביות - עדיף להשתמש ב RDBMS ו NewSQL.
- דרישה לנתונים נדיפים – נתונים המאופיינים במודל אובייקטי משתנה ופורמט נתונים מובנה וגמיש, העדיפות היא NoSQL לאחר מכן NewSQL. RDBMS פחות מומלץ בגלל הסכימה הקשיחה, יקר מאוד למימוש.
- יכולת הרחבה- הרחבה ב RDBMS כרוכה בהפצת נתונים בין צמתים רבים. כך, התחזוקה נעשת עד בלתי אפשרית. ב- NoSQL ו NewSQL מגבלה זו אינה קיימת ולכן הם קלים לתחזוקה.
- רמת ביצועים –הגורמים המכריעים הם פורמט הנתונים ומספר הפעולות המבוצעות. מבחני ביצועים הראו של NewSQL קיימים ביצועים טובים יותר מהשאר ביכולת ההרחבה ובמספר טרנזקציות המטופלות בשנייה.

באיור 8 ניתן לראות מגמת גוגל עבור שלושת המערכות. ניתן לראות שבשנים האחרונות קיימת צמיחה של NoSQL והפחתה ב- RDBMS. NewSQL התחיל להיות בצמיחה והחל משנת 2011 החל ליפול בצורה חזקה (לא הצליחו למצוא פתרון שצלח, עם תמיכה מלאה ב ACID לא ניתן להגיע לביצועים של NoSQL).



איור 8 - גוגל Trend לבסיסי נתונים RDBMS, NoSQL, NewSQL

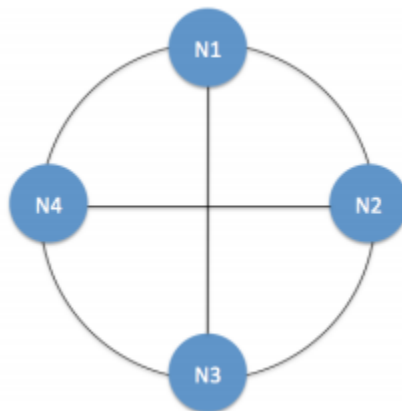
3.3.3 מנועי NoSql מרכזיים

כיום קיימים מנועי NoSQL רבים. נציין את המרכזיים [39], [37], [33], [27], [43]:

- **Google BigTable** – בסיס נתונים זה נבנה כדי לשמש כמנוע לאינדוקס הנתונים של החברה [43]. חברת גוגל פיתחה מערכת לניהול איחסון פשוט שיכול לספק גישה מהירה לנתונים המחולקים בין מכוונות רבות. BigTable מדמה אינדוקס B-Tree בו צמתי הענף והעלים מפוזרים על פני מכוונות רבות. צומת B-Tree גדלה היא ממוצלת והצמתיים מפוזרים. כך ניתן לבצע הרחבה גבוהה בין מכוונות שונות. זיהוי הנתונים ב BigTable מבוצע על פי מפתח ראשי, שם עמודה וחותרת זמן אופציונאלי. חלק גדול מהמוצרים של Google משתמשים במנוע זה להחזיקת המידע שלהם.

- **HDFS – Hadoop** מהווה תשתית לפרוייקטים של בסיסי נתונים מסוג NoSql [33]. Hadoop מתאימה לעיבודי אצווה. על מנת לעבוד עם Hadoop בזמן אמת, ניתן להשתמש ב HBase (מוסבר בסעיף 3.5.6) המצוי מעל שכבת HDFS.

- **קסנדרה (Cassandra)** – בסיס נתונים מבוזר מסוג NoSQL שנכתב במקור בפייסבוק, ועבר לקרן אפאצי [37]. הקוד פתוח, בשפת Java. ניתן להדר את קסנדרה מעל HDFS בדומה ל HBase [37]. לקסנדרה יכולות מורכבות כגון Matirialized Views ואינדוקס וכן יכולות ניהול מתקדמות יותר. בסיס הנתונים הוא מסוג multi-master database ומתאפיין בשרידות ו-Scalability גבוהים, אין לו נקודת כשל בודדת. קסנדרה מיועד, בעיקר לעבודה בזמן אמת, בסיס נתונים זה אינו משתמש בשפת SQL לצורך גישה לנתונים, הוא בעל שפה משלו (Cassandra Query Language) CQL. באיור 9 מודגמת ארכיטקטורת קסנדרה, בה הצמתים מחוברים ברשת עמית לעמית, ומופצים בצורת טבעת. כל הצמתים זהים (לעומת מערכות אחרות בהם קיימת צומת ראשית המנהלת את שאר הצמתים) לכן ההתקנה והתחזוקה קלים לשימוש.



איור 9 – ארכיטקטורת Cassandra [37]

קסנדרה התחילה והתפתחה בנפרד מ Hadoop לכן דרישות התשתית וצורת התפעול שלה שונים מ Hadoop. שימוש בקסנדרה מעל Hadoop מוסיף מורכבות לתשתית, עם זאת קיימות חברות המשתמשות בקסנדרה מעל Hadoop עבור ניתוחים אנליטיים. ב 2014, בדירוג מנועי בסיס נתונים, דורג כשני (אחרי mongoDB) מבין בסיסי נתונים NoSQL בעולם.

- MongoDB** - אחד המנועים הפופולאריים ונפוצים בתעשייה [39]. מנוע דינאמי בנוי על תפיסה של Object Oriented. מכיל תמיכה ב Java ו C#, ומאפשר שימוש בקידוד ובשאלות SQL לאחזור מידע.

בבסיס נתונים רלציוני קיימות טבלאות, הבנויות משורות ולכולן בעלות אותו מבנה. ב MongoDB עובדים עם אוסף (Collection) המורכב ממסמכים, כל אחד הוא אובייקט java script שהם אוסף של key,value. המסמכים יכולים להיות בעלי מבנה שונה. לדוגמא במסמך אחד ניתן להגדיר את ההורה ואת הילדים שלו, ובמסמך שני רק את הילדים.

שפת ניהול בסיס הנתונים היא javascript (מקבילה של sql). החיסרון של mongoDB הוא חוסר יכולת לבצע join כשצריך להביא נתונים ממספר מסמכים שונים במקביל.

MongoDB מתאים לעיבודים בזמן אמת בבסיס נתונים קטן, הפונקציונליות שלו הכי קרובה לבסיס נתונים רלציונים, קל להבנה ולשימוש, החיסרון, לא מתאים לעבודה עם טרנזקציות מרובות.]

בטבלה 2 מוצג ניתוח השוואתי בין HBase, Cassandra ו MongoDB [39],[40].

מדדי השוואה	HBASE	CASSANDRA	MongoDB
תשתית	מבוסס עמודות (רשומות עם יכולת שמירה דינמית של עמודות) וממומש מעל Hadoop. שפת יישום JAVA	מבוסס עמודות (רשומות עם יכולת שמירה דינמית של עמודות). שפת יישום JAVA	מבוסס מסמכים (בסיס נתונים בנוי מאוסף מסמכים). שפת יישום C++
זמינות	משתמש בשכפול וגיבויים של הצמתים עבור זמינות גבוהה (צומת ראשית מנהלת את שאר צמתים, ייתכן מצב של נקודת כשל בודדת)	זמינות גבוהה, אין נקודת כשל בודדת (כל הצמתים זהים, תמיד קיים גיבוי)	משתמש בשכפול וגיבויים של הצמתים עבור זמינות גבוהה (צומת ראשית מנהלת את שאר צמתים, ייתכן מצב של נקודת כשל בודדת)
עקביות	עקביות חזקה (Strong Consistency)	עקביות בסופו של דבר (Eventually Consist)	עקביות בסופו של דבר (Eventually Consist)
סכמה	סכמה משתנה (ניתן להוסיף ולהוריד עמודות באופן דינמי)	סכמה משתנה (ניתן להוסיף ולהוריד עמודות באופן דינמי)	ללא סכמה (רשומה לא חייבת להכיל מבנה של מסמך)
יכולת הרחבה	יכולת הרחבה גבוהה	יכולת הרחבה מצטברת (Incremental scalability)	יכולת הרחבה גבוהה (משתמש בשבר אוטומטי , Auto

Sharding בבסיס נתונים (עבור קלות בהרחבה)			
תמיכה מלאה באינדקס	אינדקס משני מוגבל, קיים רק בשאילתות שוויון	לא תומך באינדקס משני	אינדקס משני
השהייה נמוכה	Trades-off בין עיקביות להשהייה	השהייה נמוכה בזמן גישה לנתונים	השהייה
תומך (בעיקר בפורמט json, המסמכים נשמרים בפורמט בינארי של json)	תומך	לא תומך	תמיכה ב XML

טבלה 2: ניתוח השוואתי בין HBase, Cassandra ו-MongoDB

מתי להשתמש/לא להשתמש בכל אחד מהמנועים:

MongoDB	CASSANDRA	HBASE	
• ניתוחים אנליטיים בזמן אמת • אחסון במטמון (caching) ויכולת הרחבה גבוהה • תחלופה ל RDBMS עבור אפליקציות ווב	• האירגון מעוניין בקלות התקנה ותחזוקה ובקוד פשוט • קריאה וכתיבה אקראית גבוהה במיוחד • אין צורך באינדקס משני מרובה • קיימת דרישה בגמישות מבוזרת/רחבה של עמודות	• עבור סריקות מבוססות טווח • יישומים עם דרישת עיקביות קפדנית • יישומים עם קריאה/כתיבה מהירים ויכולת הרחבה גבוהה	להשתמש במנוע
• יישומים עם ריבוי טרנזקציות • יישומים עם דרישות של בסיס נתונים מסורתיים כגון אילוצי מפתח משני וכ"ו	• יישומים בהם קיים: בסיס נתונים רלציוני • טרנזקציות • שאילתות דינמיות על עמודות • חיפוש לפי עמודה • אינדקס משני • השהייה נמוכה	• יישומים עם טרנזקציות • יישומים בהם קיים: - צבירה, גלגול נתונים, ניתוח נתונים לפי שורות - סריקה מלאה של טבלאות	לא להשתמש במנוע

MapReduce 3.4

3.4.1 מבוא

MapReduce [25], [19] הינו מודל תכנות פשוט ורב עוצמה שפותח ע"י חברת גוגל בשפת C++, המאפשר פיתוח של יישומים הניתנים להרחבה, הרצים במקביל ומעבדים כמויות עצומות של מידע על גבי אשכולות ענק. תשתית MapReduce כוללת ממוש פונקציות **מיפוי** (map) ו**צמצום** (reduce) על ידי המשתמש, ו**ספריות** ה - MapReduce. הספריות מטפלות באופן אוטומטי בהפצת נתונים, עבודה מקבילית, איזון עומסים, עמידות בפני תקלות ובעיות נפוצות אחרות. היישום מבודד מכל הפרטים של ההפעלה. למשתמש חשופות רק מתודות המיפוי והצמצום.

3.4.2 מודל התכנות

באיור 10 מודגם שלבי MapReduce. נסקור זאת בהרחבה.

נגדיר:

פועלים (Workers): מבצעים את העבודה בפועל.

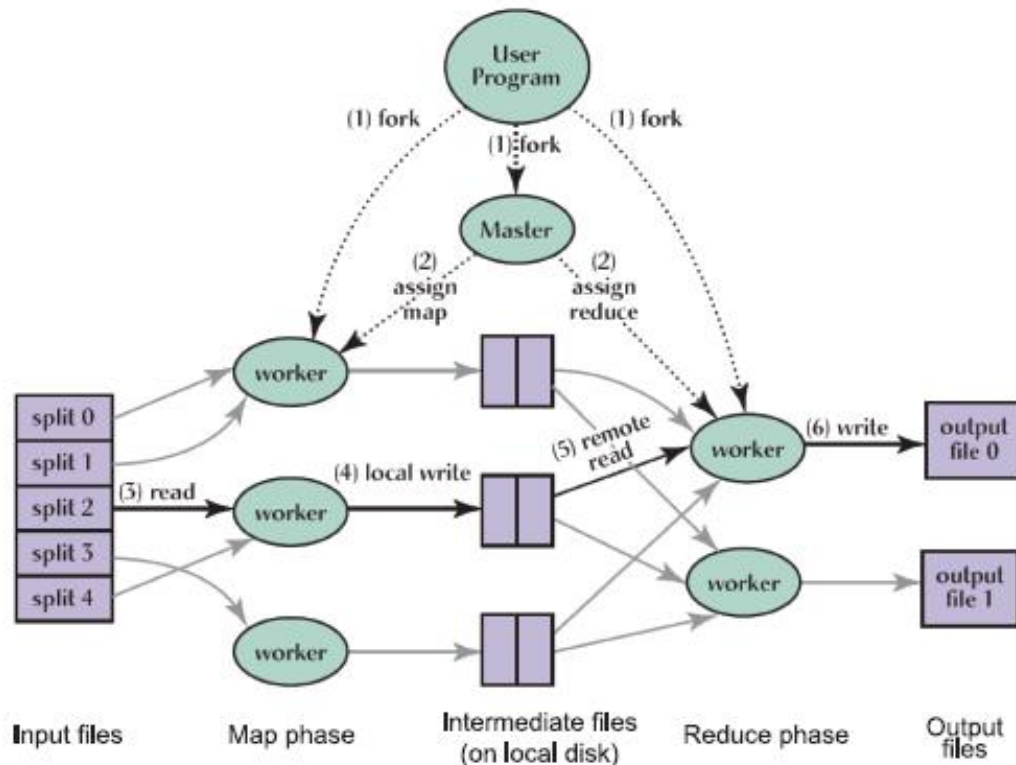
צומת ראשית (Master): אחראית על חלוקת העבודה לפועלים, קיים אחד באשכול.

הרצת MapReduce מורכבת משלושה שלבים: שלב המיפוי, שלב ביניים, ושלב הצמצום.

כעת נתבונן באיור 10. נתון קלט ספריית MapReduce המפצלת את הקלט ל - M חלקים בגודל 16-64 מגה

ביט **פיצול** (split) מודגם באיור 10 בצד שמאל (input files), ומריצה העתקים של הקלט בצמתים רבים

באשכול. קיימות M משימות מיפוי לריצה, אחד עבור כל פיצול.



איור 10 – שלבי זרימת מידע ב MapReduce [25]

נתאר את שלבי MapReduce לפי הסדר המתואר באיור 10 (ע"פ מספור):

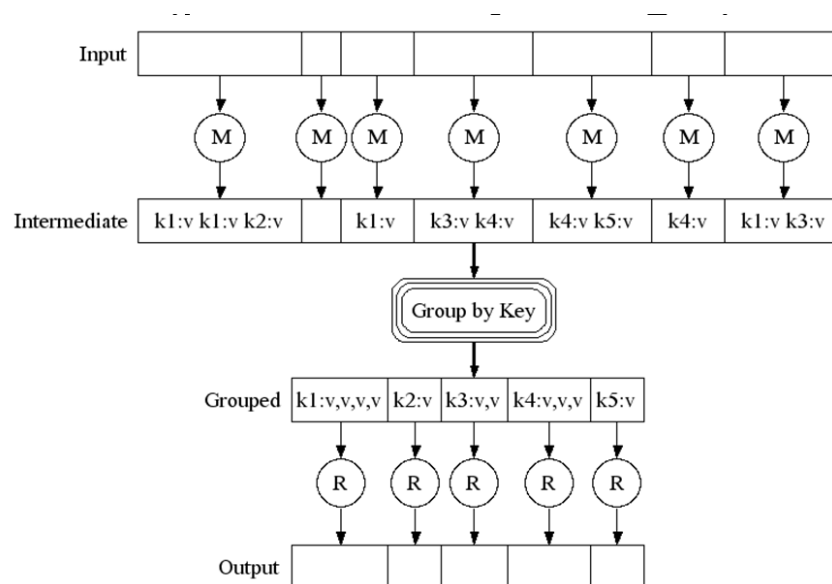
- (1) אחד העתקים נבחר להיות הצומת הראשית (Master), השאר נחשבים כפועלים (workers)
- (2) הצומת הראשית בוחרת פועלים במצב המתנה ומקצה להם משימות של מיפוי ו/או צמצום.
- (3) פועלים עם משימות מיפוי, מקבלים את הקלט, מעבדים את התוכן, יוצרים זוגות של <ערך, מפתח> ומעבירים את הקלט לפונקציית מיפוי של המשתמש.
- (4) ערכי הביניים <ערך, מפתח> נשמרים בזיכרון, ומעת לעת נכתבים לדיסק מקומי המחולק ל R מחיצות (פונקציית מחיצה של המשתמש, partition, יוצרת R מחיצות).
- * המיקום של הזוגות נשלח לצומת הראשית האחראית להעביר את המיקום לפועלים עם משימות צימצום.
- * קיימים R משימות צימצום כמספר המחיצות
- * ספריות MapReduce ממיינות את ערכי הביניים (בעזרת מפתחות הביניים), כך שכל המופעים של אותו מפתח מקובצים יחד (המיון נועד עבור קיבוץ של הערכים לפי אותו מפתח).
- (5) פועלים עם משימות צימצום, מקבלים את המיקום של תוצאות הביניים מהצומת הראשית, וקוראים את ערכי הביניים מהמחיצה המתאימה מדיסק מקומי של פועלים עם משימות מיפוי (הפועלים מעבירים את המפתח ואת אוסף כל הערכים עבור אותו מפתח, לפונקציית צימצום של המשתמש)

(6) פונקציית צימצום רצה על הערכים ויוצרת את הפלט (output file)
 * הפלט של פונקציית צימצום מצורפת לפלט הסופי של המחיצה.

כאשר כל משימות המיפוי והצמצום מסתיימות בהצלחה, צומת ראשית מעירה את תוכנת המשתמש ומחזירה את השליטה לקוד המשתמש.

תשתית MapReduce

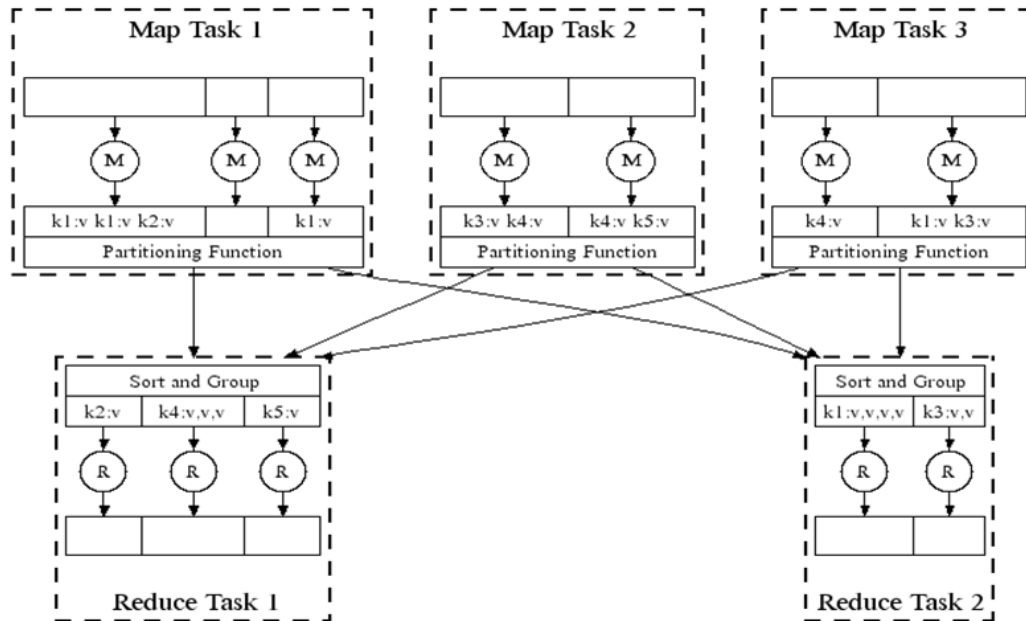
באיור 11 ניתן לראות את תשתית ה MapReduce [38]. עבור כל קלט (input), שלב המיפוי (M) מחלק את הקלט לרשומות של מפתח וערך (k,v) , מפעיל את פונקציית המיפוי של המשתמש, ומעביר את הנתונים לשכבת הביניים (Intermediate). ספריית MapReduce מבצעת מיון וקיבוץ של הערכים לפי מפתח (פעולת Grouped by Key , בונה מערך של ערכים עבור כל מפתח, כדי להפחית את כמות הנתונים הדרושה להעברה ברשת. פונקציית הקיבוץ רצה על אותו מחשב בו רצה משימת המיפוי). בשלב האחרון מופעל שלב הצימצום (R) , המקבל כקלט את המפתח ומערך הערכים של אותו המפתח, ומפעיל את פונקציית הצמצום של המשתמש, ובונה את הפלט (Output). המשתמש צריך להגדיר את פונקציית מיפוי (איך לחלק את הבעיה הגדולה) ואת פונקציית צימצום (איך לאחד את הערכים), המיון והקיבוץ מתבצעות בספריות של ה MapReduce.



איור 11 – תשתית MapReduce [38]

עיבוד מקבילי ב - MapReduce

באיור 12 ניתן לראות את הריצה המקבילית של ה MapReduce [38]. הקלט מחולק ל M חלקים, פונקציית מיפוי רצה במקביל על צמתים שונים וממפה את הקלט למפתח וערך. תהליך מיון וקיבוץ רץ במקביל על כל התוצאות המיפוי, ומצרף את הערכים לפי המפתח, בהמשך מופעלת פונקציית הצימצום ובונה את הפלט. MapReduce לא חייב לרוץ על פני מכונות רבות, קיימים מימושים שונים ל MapReduce לדוגמה Phoenix הרצים על מכונות בודדות עם הרבה מעבדים.



איור 12- עיבוד מקבילי ב MapReduce [38]

נדגים MapReduce :

נתון קלט אוסף של קבצים, ופלט מספר המופעים של כל מילה.

באיור 13 ניתן לראות את פונקציית מיפוי וצימצום המוגדרות על ידי המשתמש. פונקציית מיפוי מקבלת

כקלט שם ותוכן של המסמך ועבור כל מילה במסמך מצרפת ערך 1.

פונקציית צימצום מקבלת מפתח (מילה) ורשימה המכילה את מספר מופעים של המילה, ומחזירה את מספר הכולל של המופעים.

עיבוד : יתבצעו שלוש פעולות : מיפוי, קיבוץ, צמצום.

שלב מיפוי (Map) - מחלק ל <ערך, מפתח>, כל מילה היא מפתח, ועבור כל מילה מצרף ערך 1 (מילה הופיעה פעם אחת בקובץ).

צירוף (Combine) - מתבצע מיון וקיבוץ של הערכים, מצרף לכל מפתח את כל ההערכים באותו הקובץ (מתבצע אוטומטית בספריות של MapReduce). הקיבוץ מתבצע בשלב המיפוי (לשיפור ביצועים).
שלב צמצום (Reduce) - מצרף לכל מפתח את כל ההערכים בין כל הקבצים.

```
/* key: document name
 * value: document contents*/

map(String key, String value){
  for each word w in value:
    emitIntermediate(w, "1");}

/* key: a word
 * values: list of counts for the word*/

reduce (String key, Iterator values){
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  emit(result);}
```

איור 13 – פונקציות מיפוי וצמצום המוגדרות על ידי המשתמש לספירת מילים [25]

כעת נדגים את תוצרי השלבים של MapReduce בדוגמא הנתונה :

file1: "hello world hello moon"

file2: "goodbye world goodnight moon"

MAP

First map:

```
< hello, 1 >  
< world, 1 >  
< hello, 1 >  
< moon, 1 >
```

Second map:

```
< goodbye, 1 >  
< world, 1 >  
< goodnight, 1 >  
< moon, 1 >
```

COMBINE

First map:

```
< moon, 1 >  
< world, 1 >  
< hello, 2 >
```

Second map:

```
< goodbye, 1 >  
< world, 1 >  
< goodnight, 1 >  
< moon, 1 >
```

REDUCE

```
< goodbye, 1 >  
< goodnight, 1 >  
< moon, 2 >  
< world, 2 >  
< hello, 2 >
```

3.4.3 MapReduce-יתרונות והסרונות

יתרונות

MapReduce מודל פשוט ויעיל לחישוב צבירות. היתרונות העיקריים של MapReduce בעיבוד נתונים [29] :

- קל לשימוש- המתכנת מגדיר את פונקציות המיפוי והצמצום בלבד. אין צורך להגדיר את ההפצה הפיזית על פני הצמתים, וכן הפעולות מתבצעות בספריות ה MapReduce .
- גמישות –אינו תלוי במבנה הנתונים והסכמה. מתכנת יכול להתמודד עם נתונים לא סדירים וללא מבנה מסוים.
- אי תלות באיחסון – מצוי מעל שכבות האיחסון. לכן, ניתן לעבוד עם שכבות אחסון שונות כגון Big Table.
- עמידות בפני תקלות - MapReduce מתגבר על תקלות על ידי ביצוע מחדש של משימות שנפלו, ועל ידי תהליכי גיבוי [25]. מעת לעת, הצומת הראשית (Master) מבצעת בדיקה האם הפועל (worker) חי (על ידי ping). אם לא מתקבלת תשובה, הפועל מסומן בסטאטוס נכשל וצומת אחרת מבצעת את המשימה. קיימת התנהגות שונה עבור משימות מיפוי וצמצום.
 - פועלים עם משימות מיפוי, שומרים את תוצאות הביניים לוקלית במחשב. אם פועל "מת", כל המשימות שהפועל הצליח לסיים מורצות מחדש בצמתים אחרים.
 - פועלים עם משימות צמצום, שומרים את הפלט גלובלית במערכות מבוזרות. אם פועל "מת", רק המשימה בה הפועל נכשל מבוצעת מחדש.
- דרך הנוספת להתגבר על תקלות ולשפר ביצועים היא בעזרת תהליכי גיבוי. לפעמים משימה יכולה לרוץ זמן רב על המכונה (בגלל טעויות או מכונה עסוקה בדברים אחרים), כאשר פעילות ה MapReduce קרובה לסיום, הצומת הראשית מתזמנת תהליכי גיבוי עבור המשימות בתהליך סיום. המשימה מסומנת בסטאטוס הושלם, כאשר המקור או הגיבוי מסתיים.
- גוגל דיווחה, MapReduce יכולה להמשיך לעבוד גם אם יהיה לה 1.2 כישלונות בממוצע, עבור משימת פיתוח בגוגל.
- מקביליות – המשימות מורצות במקביל ולכן קיים חסכון משמעותי בזמן ריצה.
- יכולת הרחבה גבוהה- ניתן להרחיב את הצמתים בזמן ריצה. כבר ב 2008 יאהו דיווחה, יכולת הרחבה גבוהה יותר מ 4000 צמתים.

חסרונות

- חסרונות של MapReduce יסקרו בהשוואה ל DBMS [29].
- העדר תמיכה בשפות גבוהות - אינה תומכת בשפות גבוהות כגון SQL ב DBMS ואופטימיזציה של שאילתות. יש להוסיף לכך קוד בפונקציות Map ו Reduce.

- העדר תמיכה בסכמות – MapReduce אינה תלויה בסכמה ואינדקס. MapReduce יכולה להתחיל לעבוד מיידית עם קליטת הקלט ובכך מאבדת את היתרונות של מודל נתונים. MapReduce מפענחת כל פריט בקלט, והופכת אותו לאובייקט נתונים עבור עיבוד נתונים, כך תחול ירידה בביצועים
- העדר גמישות בזרימת מידע- מודל ה MapReduce מוגבל, חושף רק את הפונקציות מיפוי וצמצום עם פורמט אחד של נתוני קלט (זוג מפתח, ערך). זרימת מידע במודל נועדה במקור לקרוא סוג קלט יחיד ולייצר סוג פלט יחיד (רק זוגות מפתח וערך). כאשר קיימת זרימה אחרת של נתונים, נוצרים בעיות בשל ההגבלה. אלגוריתמים הדורשים סוגי קלטים מרובים לא יכולים לרוץ ב MapReduce) את הקוד של האלגוריתמים יש לממש בפונקציות Map ו Reduce).
- הגבלה בפעולות Join – צריך להתאים את הקוד לפילטור ו projection משגורם לבעיות בתחזוקה ושימוש חוזר של קוד.
- הגבלה בלולאות - אלגוריתמים עם לולאות צריכים מידע גלובלי של state בזמן ריצה (לדעת מתי לולאה נגמרת), MapReduce לא מתייחס ל state וקוראת את אתו המידע באיטרציות. בכל איטרציה הנתונים נטענים ומעובדים מחדש למרות שייתכן והמידע לא משתנה, וזה גורם לבזבוז קלט/פלט וזמן מעבד.
- ביצועים :
 - חסימה – מיפוי וצמצום הן פעולות חסומות. מעבר לשלב צמצום לא מתבצע עד שכל המשימות של השלב מיפוי הסתיימו (בגלל הצורך בצירוף ומיון נתונים בשלב הביניים), משגורם לירידה בביצועים (מקביליות בצינור לא מנוצלת) ולא מאפשר עיבוד מקוון.
 - אופטימיזציה לתוכניות - אין אופטימיזציה לתוכניות (לא מייעלת תוכניות כמו DBMS המנסה למזער את כמות הנתונים הרצים בצמתים).
 - אופטימיזציה לקלט פלט -פעולות MapReduce לא תמיד מותאמות לפעולות קלט ופלט בצורה יעילה, המטרה הראשית של ה MapReduce היא עמידות מבפני תקלות והרחבה לא מוגבלת.

נציג דרכים לשיפור חסרונות של MapReduce שצוינו בסעיף הקודם [29], [19] :

- טיפול בהעדר **תמיכה בשפות גבוהות** – שימוש בשפות התומכות בשפת שאילתות מעל MapReduce כגון Hive, Pig Latin. שפת השאילתות לא תלויה בלוגיקה עיסקית וניתן לבצע שימוש חוזר ואופטימיזציה לשאילתה. בנוסף שפת שאילתות מעל MapReduce, אוטומטית בונות את הפונקציות מיפוי וצימצום של ה MapReduce ובכך מקלה על הפיתוח.
 - טיפול בהעדר **תמיכה בסכמות** – ניתן להשתמש בפורמטים כגון XML, JSON לבדיקת שלמות נתונים, אך נתונים המכילים נתוני סכמה עשויים לגדול. לפיכך, ניתן לבצע דחיסה של נתונים.
 - טיפול בהעדר **גמישות בזרימת מידע**
 - **טיפול בלולאות** - מערכת HaLoop תומכת בלולאות. HaLoop פותרת את הבעיה של האיטרציות על ידי שמירת מידע באיטרציה ראשונה, ושימוש במידע בשאר האיטרציות.
 - **טיפול בפעולות פגולות join** - פעולות איחוד דורשת יותר משני קלטים, MapReduce תומך בקלט אחד. ניתן לפתור את הבעיה בשתי דרכים שונות, איחוד בצד מיפוי או איחוד בצד צמצום.
- איחוד בצד של מיפוי: איחוד העובד בדומה ל sort merge join ב DBMS. איחוד מתבצע בשני שלבים. בשלב הראשון מחלקים את שני הקלטים למחיצות וממיינים לפי המפתח של האיחוד. בשלב השני ה Mappers קוראים את הקלטים וממזגים אותם.
- איחוד בשידור היא שיטה אחרת של איחוד בצד מיפוי המתאימה למקרה בו גודל אחד היחסים קטן. היחס הקטן משודר לכל mapper ונשמר בזיכרון, בדרך זו חוסכים בפעולות קלט/פלט של העברה ומיון בשני היחסים. איחוד בשידור משתמש בטבלאות גיבוב לאיחסון היחס הקטן, ולמציאת התאמות ביחס השני.
- איחוד בצד צמצום: משתמשים בשיטת החלוקה מחדש. Mapper מסמן רשומות בשני היחסים כדי לזהות מאיזה יחס הגיע הרשומה. במהלך העירבוב, רשומות עם אותו ערך מפתח מעותקות לאותו reducer. בשלב האחרון כל reducer מבצע את האיחוד של הרשומות לפי המפתח (דומה לפעולת hash join ב DBMS).
- **טיפול בשיפור רמת הביצועים**
 - **טיפול בחסימה** - MapReduce Online היא המצאה התומכת בצבירה מקוונת על ידי שינוי ארכטקטורה. בסיום הפעולה של פונקציית map, Mappers מעת לעת מאחסנים לוקלית את הנתונים אצל ה reducers, כך ש reducers יוכלו לצבור ולא לחכות עד שתהליך כולו של ה map יסתיים. בנוסף, ניתן לוותר על המיון על ידי שימוש ב Hash Table (פונקציות hash היוצרת buckets), תהליך המיון משפיע מאוד על ביצועים. בעזרת hash table אין צורך בקיבוצ, כל bucket שומר את כל הערכים לפי מפתח (פעולת

המיון נועדה לעזור בקיבוץ של כל הערכים תחת אותו מפתח). ה reducers לסוף סוכמים את הערכים מכל bucket.

- **אופטימיזציה לתוכניות - רוב תוכניות ה MapReduce נכתבות עבור ניתוח נתונים,** ולוקח להן זמן רב לסיום. במקרים אלו ניתן לבצע אופטימיזציה לתוכניות. אחת הדרכים, למצוא פרמטרים אופטימלים לקלט נתון, על ידי ריצות ספקולטיביות על מדגם של נתונים. דרך שניה, לבחון את הקוד ללא ריצה, על פי החוקים לבנות עץ B tree, ובהתאם לעץ לפרוס את הנתונים. בנוסף ניתן להשתמש בטכיקות של דחיסה לצמצום קלט פלט.
- **אופטימיזציה לקלט פלט - קיימות גישות המפחיתות את פעולות קלט/פלט על ידי שימוש במבנה אינדקסי ודחיסת מידע.**
- **תזמון – הצומת "מהירה", המסיימת את המשימה שלה במהירות, תקבל יותר משימות.** צומת "איטית" שלוקח לה זמן רב להריץ משימה, תורץ מחדש על צומת פנויה אחרת. שיטה זו אינה מתאימה לסביבות בהן לצמתים קיים כוח מיחשוב שונה, משימה שרצה מהר, ומורצת על צומת עם כח מיחשוב קטן עדיין תסתיים אחרונה.
- **חיסכון באנרגיה - עלות האנרגיה במרכזי מחשוב היא 23% מכלל העלויות, ולכן חשוב לחסוך באנרגיה.**

יש לבצע תכנון נכון של צריכת אנרגיה בין הצמתים הלא פעילים. באחת מבין שתי הגישות: **Covering-Set**: ריצות בהן קיים ניצול נמוך של צמתים, אפשר לוותר על חלק מהצמתים ולכבות אותם. את המידע של בלוק נתונים מעתיקים לכל שאר הצמתים הפעילים (עדיין ניתן לגשת לכל הנתונים בגלל העתק מידע), עלולה להיווצר בעיה רק במקרה שהייתה נפילה מרובה של צמתים פעילים.

All-In strategy: עבודות מיפוי נכנסות לתור, ומחכות עד שהתור יתמלא לסף מסויים. כאשר התור מתמלא, העבודות מורצות. לאחר סיום ריצה, מכבים את כל הצמתים, עד שהתור מתמלא שוב.

שתי השיטות פוגעות בביצועים וחוסכות באנרגיה.
- **מערכות היברידיות – שילוב מערכות.** לדוגמא HadoopDB מערכת היברידית המחברת בין שתי מערכות DBMS ו MapReduce. שאליות כתובות בשפת SQL ומפוזרות בין הצמתים על ידי MapReduce.

3.4.5 שכבות תוכנה הקיימות מעל MapReduce

MapReduce מוגבל, מכיל פורמט אחד של נתוני קלט ולרוב קוד מותאם אישית עבור פעולות נפוצות. מתכנתים רבים מעדיפים להשתמש בשפת SQL להגדרת משימות תוך השארת אופטימיזציה וביצוע למנוע backend. נציג ממשקים הנמצאים בשכבה מעל ה MapReduce [19]:

- **Sawzall** – שפת סקריפטים. המשמשת בגוגל לעיבוד של כמויות גדולות של רשומות לוג. שרתי לוגים מאוחסנים במערכת קבצים של גוגל GFS (Google File System) המחולקים למחיצות בדיסקים רבים. על מנת לבצע חישובים על לוגים צריך להדר תוכניות MapReduce היכולות לצרוך זמן רב. לפיכך, בעזרת שפת הסקריפטים, הסקריפט רץ בשלב מיפוי ומעביר פלט של ערכים לטבלה. בשלב צמצום יאספו הפלטים מהטבלאות המרובות לקבוצה אחת של טבלאות.
- **Pig** – שכבת תוכנה שמטרתה לפשט כתיבת תוכניות MapReduce, פותחה במקור על ידי yahoo. Pig מורכבת משפת Pig Latin, המגדירה תוכניות ריצה שמתורגמות אוטומטית לתוכניות MapReduce. השפה כתובה כרצף של צעדים, כאשר כל צעד מייצג שינוי יחיד בנתונים. באיור 14 ניתן לראות שאילתת SQL ומקבילה שלה בשפת Pig Latin

<u>SQL</u>	<u>Pig Latin</u>
<pre>SELECT category, AVG(pagerank) FROM urls WHERE pagerank > 0.2 GROUP BY category HAVING COUNT(*) > 10⁶</pre>	<pre>good_urls = FILTER urls BY pagerank > 0.2; groups = GROUP good_urls BY category; big_groups = FILTER groups BY COUNT(good_urls)>10⁶; output = FOREACH big_groups GENERATE category, AVG(good_urls.pagerank);</pre>

איור 14 – שאילתת SQL ומקבילה שלה ב Pig Latin [19]

Pig שימושית לכתיבת תוכניות ETL (תהליך מבוסס תוכנה באמצעותו מועברים נתונים ממערכות תפעוליות למחסן נתונים) מעל Hadoop ומקטינה את הצורך לכתוב בשפת Java.

- **HBase** – יפורט בסעיף 3.5.6
- **Hive** – תשתית עבור מחסן מידע מעל שכבת Hadoop. שימוש ב HBase אינו אינטואיטיבי, פייסבוק פיתחו את ממשק Hive מעל HBase עם שפה דמויית SQL בשם HiveQL המתורגמת בזמן הריצה לסט תוכניות MapReduce. המטרה העיקרית של הפרוייקט, היא להביא את המושגים העיקריים המוכרים במסדי נתונים רלציוניים לעולם מובנה של Hadoop, תוך שמירה על הרחבה וגמישות ש-

Hadoop מספקת. Hive תומך בניתוח מערכות גדולות המאוחסנות ב HDFS, השפה מקלה על כלי BI לתחקר מידע שנמצא ב hadoop ומאפשרת דוחות פשוטים. לדוגמא, ספירת מילים המוצגת באיור 13, ניתן לבטא בעזרת HiveQL המוצג באיור 15.

```
FROM (
  MAP doctext USING 'python wc_mapper.py' AS (word, cnt)
  FROM docs
  CLUSTER BY word
) a
REDUCE word, cnt USING 'python wc_reduce.py';
```

איור 15 – שאילתת HiveQL לספירת מילים [19]

שאילתת משנה, Map מציין פונקציית מיפוי של משתמש, שנקראת wc_mapper.py, ההופך את הקלט לערכים של מפתח וערך (word,cnt), ופונקציית צמצום wc_reduce.py המופעלת את עמודות הפלט של שאילתת המשנה (word,cnt).

- **Tenzing** – בדומה ל Hive, פותח עבור גוגל, ועובד מעל מערכת קבצים של גוגל. נועד עבור ניתוחים גדולים, תומך בשפת SQL, ומתרגם שאילתות בזמן ריצה לסט תוכניות MapReduce.
- **Jaql** – שפת שאילתות שעוצבה עבור JSON (Javascript Object Notation), אובייקטים פורמט טקסטואלי פשוט המיועד לייצוג מבני מידע ושכבות). משמש עבור ניתוח נתונים בקנה מידה גדול. זוהי שפה המשכתבת שאילתות ברמה גבוהה, ומתאימה אותם לשאילתות ב Hadoop MapReduce. Jaql מסוגל לעבד נתונים ללא סכמה או עם סכמה חלקית. י.ב.מ משתמש בו כשפת עיבוד מרכזית לחבילות תוכנה עם Hadoop.

3.5.1 מבוא

Hadoop הוא קוד פתוח של קרן התוכנה אפאצי, מבוסס Java. Hadoop הוא אחד הפיתרונות הבולטים בארגונים לטיפול ב - Big Data בענן, המספק תשתית לאחסון ועיבוד מקבילי על גבי אשכולי ענק של עד אלפי שרתים. באמצעות הכלים וטכנולוגיות שהפלטפורמה מספקת, ניתן לקחת פעולה כבדה (מבחינת זמן וכמות הקלט/פלט הנדרש) ולפזר אותה על כמות גדולה של מחשבים חלשים בכדי להשיג את התוצאה במהירות. השימוש הבסיסי של Hadoop הוא אחסון המידע רב באופן קל ופשוט, כך שניתן לנצל אותו לצרכי BI.

פלטפורמת Hadoop מתחלקת לשני חלקים עיקריים, ניהול מידע ע"י אחסונו במערכת קבצים מבוזרת בין שרתים פיזיים שמכונה **HDFS** (Hadoop File System) ומנגנון חישוב מקבילי **Hadoop MapReduce**.

באמצעות HDFS נקבל חלוקה של קבצים גדולים לקבצים קטנים יותר המפוזרים בין השרתים השונים עבור קריאה מקבילית של המידע בין השרתים. קבצים נשמרים מקומית על השרתים, ולרוב לא משותפים בין כל השרתים. HDFS הוא התשתית לפרויקטים של בסיסי נתונים מסוג NoSQL ומהווה תחליף של בסיס נתונים מסורתי.

Hadoop MapReduce מספקת תשתית מבוזרת לעיבוד מקבילי. התשתית אחראית על עיבוד נתונים וניהול משאבי אשכול. התשתית מחלקת משימות של הקלט על ידי שימוש ב HDFS ומפזרת הפעילות על כלל השרתים. התשתית גם יודעת להתמודד עם שגיאות בזמן הריצה כגון נפילת שרתים או בעיות רשת.

בגרסאות החדשות של Hadoop, נוספה שכבה Yarn, האחראית על ניהול משאבים במקום ה MapReduce. הוספת Yarn פותרת את מגבלותיו של Hadoop ומאפשרת ניצול טוב יותר של אשכול.

רכיבי האשכול ב- Hadoop הם :

- NameNode – מנהל את המרחב, metadata של קבצים, ובקרת גישה. קיים NameNode אחד באשכול.
- Secondary NameNode – מרענן את ה metadata ב NameNode (קורא את השינויים במערכת הקבצים), קיים אחד באשכול.
- JobTracker - שירות דימון, לשליחה ואיתור של עבודות MapReduce ב Hadoop. קיים אחד באשכול.
- TaskTracker - דימון המקבל עבודות (מיפוי, צמצום, עירבוב) מ JobTracker ומבצע אותן.
- DataNode – מחזיק את הנתונים של מערכת קבצים. כותב וקורא בלוקים מהדיסק לפי הוראות שמקבל מ-NameNode.

בסעיפים 3.5.2, 3.5.3 יפורטו בהרחבה סעיפים אלו.

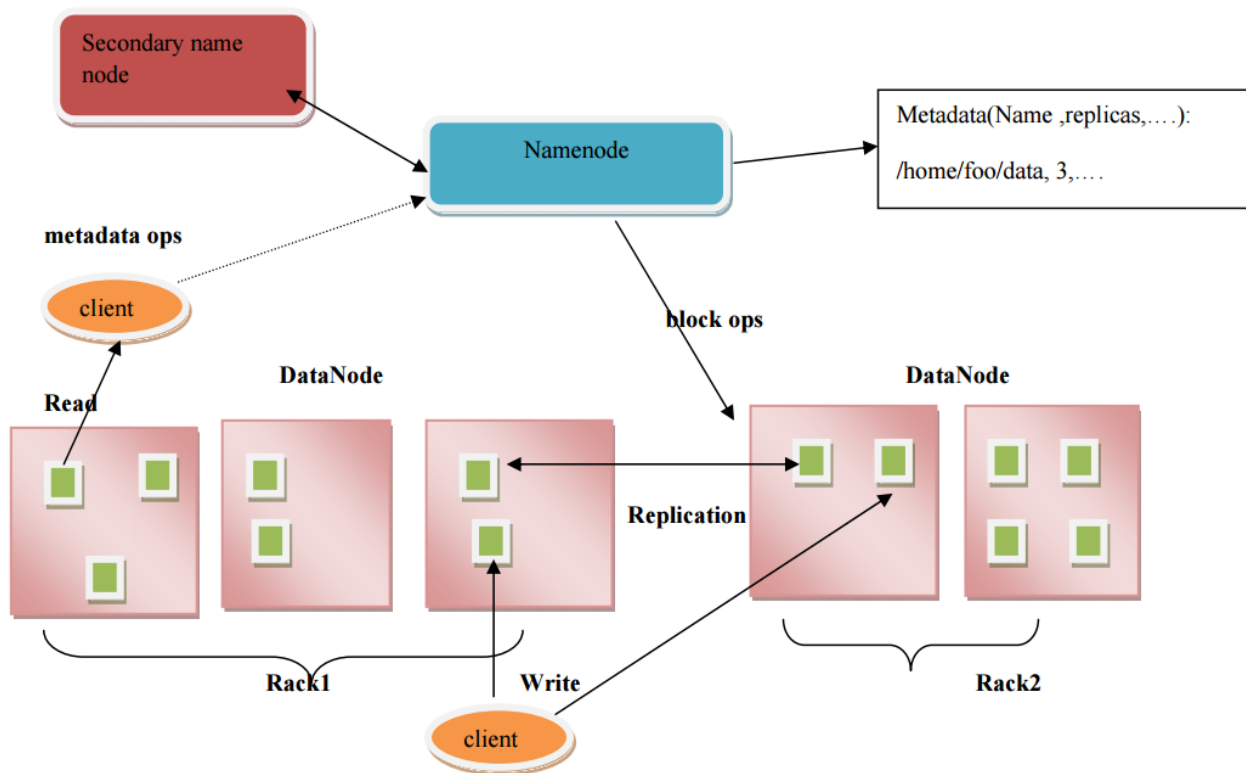
HDFS 3.5.2

HDFS היא מערכת לאחסון קבצים מבוססת בעלת נפחי מידע גדולים (petabytes), זמינות מידע גבוהה, תמיכה גבוהה בתעבורת מידע, ושילוב טוב עם תשתית ה MapReduce. יחידת האחסון הבסיסית ב HDFS היא בלוק כאשר ברירת המחדל של גודל בלוק הוא 64 מגה בית. קבצים מחולקים לבלוקים שווים (פרט לבלוק האחרון) ומפוזרים על פני אשכול של מכונות. על מנת להתגבר על בעיית זמינות של בלוק, כל בלוק משוכפל שלוש פעמים על גבי מכונות שונות (במקרה ובלוק לא זמין, המערכת אוטומטית תנתב את המשתמש לבלוק אחר) [16], [17], [20].

מבנה העבודה של HDFS הוא בשיטת מנהל-פועל. כל אשכול ב HDFS מורכב משני סוגי צמתים המנהל (NameNode) והפועל (DataNode). שרת אחד באשכול מוקדש להיות רכיב **המנהל** שאחראי לניהול כל האשכול. מספר שרתים, מוקדשים להיות רכיבי **הפועל** האחראים על הגישה למידע. הרכיבים מתקשרים בינם לבין עצמם תמידית באמצעות פרוטוקול TCP. אשכול יכול להיות מפוזר על גבי מספר אתרים פיזיים שונים, במקומות שונים בעולם.

ארכיטקטורה

באיור 16 ניתן לראות את ארכיטקטורה של HDFS. אשכול המכיל את רכיב המנהל (NameNode), רכיב המנהל המשני (Secondary nameNode), רכיבי פועלים (Data Node) ורכיב הלקוח (Client). רכיב הפועל, מורכב מבלוקים, כאשר העתק של אותו בלוק (Replication) נמצא ברכיבי פועלים אחרים על גבי התקנות שונות (Rack). רכיב המנהל המשני, מרענן את הנתונים ברכיב המנהל. רכיב **הלקוח** (אחראי על תקשורת עם העולם מחוץ ל HDFS) ניגש לרכיב המנהל לקבלת רשימת הפועלים הרלוונטים ומבצע את פעולת כתיבה (write) וקריאה (read) מול רכיבי הפועלים. הרכיבים מתקשרים בינם לבין עצמם תמידית באמצעות פרוטוקול TCP.



איור 16 – ארכטקטורת HDFS [16]

NamedNode - אחראי על חלוקת העבודה לפועלים, קביעת הרשאות גישה לקבצים, וויסות גישה על מנת למנוע מצבי עומס, שמירת ה Metadata על הקבצים הנשמרים ברכיבי הפועלים (מיקום הבלוקים, כמות העתקות לכל קובץ, הרשאות, וכ"ד) ותקשורת עם רכיב הלקוח (Client). רכיב המנהל יודע אילו בלוקים מרכיבים כל קובץ ובאיזה רכיב פועל הם קיימים.

Secondary nameNode – רכיב מנהל המשני, תפקידו לרענן את ה Metadata שנשמר ברכיב המנהל. המנהל המשני מתחבר אחת לתקופה המוגדרת מראש, בדרך כלל שעה, אל רכיב המנהל, מעתיק ממנו את ה Metadata על הקבצים המאוחסנים ברכיבי הפועלים, מרענן אותם, ומעלה אותם בחזרה אל רכיב המנהל הראשי. עותק של תוצאת הפעולה נשמר ברכיב המנהל המשני. במידה ורכיב המנהל נופל, ניתן לבנות אותו מחדש בעזרת המידע שקיים ברכיב מנהל משני.

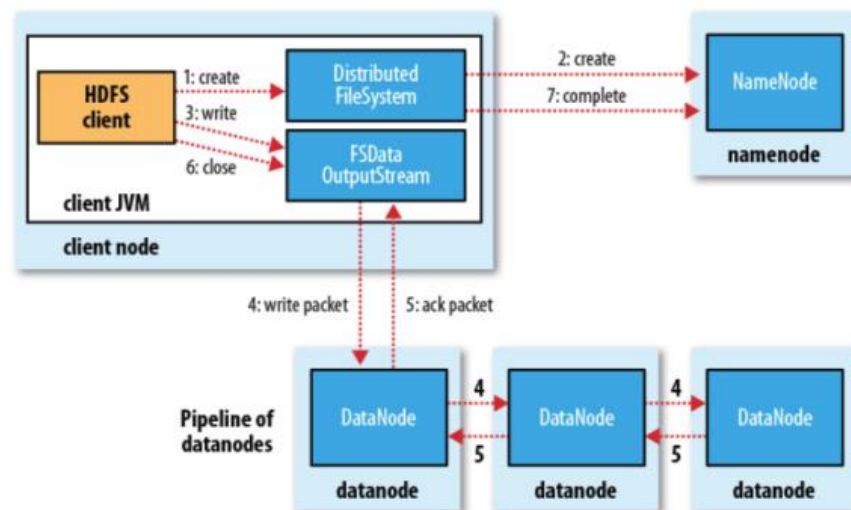
DataNode – רכיב הפועל, מכיל את הקבצים השמורים בבלוקים על גבי הדיסק, הוא כותב וקורא בלוקים מהדיסק לפי הוראות שהוא מקבל מרכיב המנהל. במהלך העבודה השוטפת מתקשר רכיב הפועל עם רכיב המנהל (בדרך כלל כל שלוש שניות) כדי להודיע שהוא חי, ומדי פעם מעביר מידע על נתוני הבלוקים. דוחות בלוקים מאפשרים לרכיב המנהל לוודא שבתוך האשכול אכן נמצאים שלושת העותקים של הקבצים (על מנת לשמור על זמינות מידע במקרה של כשל חומרת).

Client - רכיב הלקוח אחראי על תקשורת עם העולם מחוץ ל HDFS. כאשר אפליקציה כלשהי מעוניינת לגשת למידע ב HDFS, רכיב הלקוח פונה אל רכיב המנהל ומבקש את רשימת רכיבי הפועלים הרלוונטיים. רכיב המנהל יספק את הרשימה הטובה ביותר מבחינת ביצועים (בדרך כלל מרכיב פועל הקרוב ביותר מכל העתקים). לאחר מכן פונה רכיב הלקוח ישירות אל רכיבי הפועלים ברשימה שסופקה לו, ומבצע מולם את הפעולה הנדרשת (read, write).

מודעות להתקנות (Rack) – HDFS מספקת מודעות להתקנות. כל צומת יודעת באיזה התקנה התקינו אותה (לפי שדה rack_id המכיל מספר ייחודי של ההתקנה). בדרך כלל אשכולות גדולים מאורגנים על פני מספר התקנות. תעבורת רשת בין צמתים שונים באותה התקנה יעילה יותר מתעבורת רשת בין צמתים בהתקנות שונות. רכיב המנהל, מנסה למקם את העתקים של בלוקים במספר התקנות שונות, עבור עמידות בפני תקלות. למנהל מערכת קיימת אפשרות להחליט לאיזה התקנה כל צומת שייך (על ידי שינוי השדה rack_id).

פעולת כתיבה ל HDFS

באיור 17 מודגמים שלבי יצירת קובץ חדש ב HDFS. נסקור שלבים אלו.



איור 17 - תרשים כתיבת קובץ ב HDFS [16]

תאור התהליך על פי המספור הקיים באיור 17 :

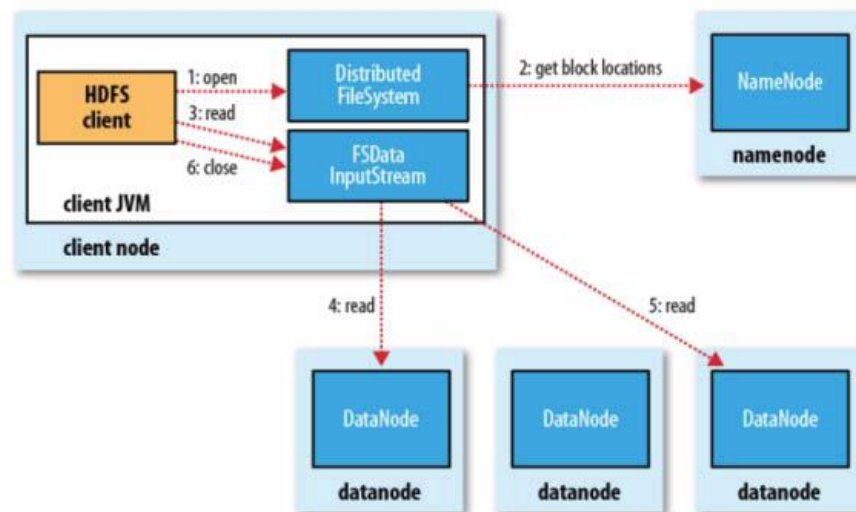
1. רכיב הלקוח מייצר את הקובץ שיש לכתוב לדיסק.
2. רכיב לקוח פונה אל רכיב המנהל עם פרטי הקובץ, ללא מידע שקשור לכמות הבלוקים הנדרשת בשלב זה. רכיב המנהל מבצע מספק בדיקות, בינהן בדיקת הרשאות, זמינות מקום פנוי ובדיקת כפילויות.

בהנחה שהבדיקות מסתיימות בהצלחה, רכיב המנהל מחזיר לרכיב הלקוח אובייקט המכונה `FSDDataOutputStream`. אובייקט זה מכיל את רשימת רכיבי הפועלים שיש לכתוב אליהם את המידע וכיצד לגשת אליהם.

3. רכיב הלקוח מתחיל לכתוב את המידע הנדרש אל האובייקט שהוחזר לו. האובייקט דואג לקבל את המידע, לפצל אותו לבלוקים ולדאוג שתבוצע פעולת השכפול הנדרשת בין רכיבי הפועלים. לקוח צריך להזרים את המידע אל אובייקט `FSDDataOutputStream`.
4. רכיב הפועל הראשון מקבל את המידע וכותב אותו לדיסק, התהליך חוזר חלילה עבור רכיב הפועל השני והשלישי (בהנחה שכמות השכפולים היא ברירת המחדל של שלושה).
5. אובייקט `FSDDataOutputStream` פונה לכל רכיב פועל ומבקש אישור לכתיבת המידע. לאחר שהאישור מתקבל מכל הרכיבים המשתתפים בפעולת הכתיבה (קבלת `ack`), ניתן להמשיך.
6. מתבצעת פעולת סגירה לאובייקט `FSDDataOutputStream`.
7. רכיב המנהל מקבל אישור שהפעולה הסתיימה בהצלחה. רכיב המנהל, מחזיר הודעת הצלחה לרכיב השרת

פעולת קריאה מ HDFS

באיור 18 מודגמים שלבי קריאת קובץ מ HDFS. נסקור שלבים אלו.



איור 18 - תרשים קריאת קובץ ב HDFS [16]

תאור התהליך על פי המספור הקיים באיור 18 :

1. רכיב הלקוח פותח קובץ ריק.
2. רכיב הלקוח פונה אל רכיב המנהל ומבקש ממנו רשימה של רכיבי פועלים עליהם נמצא המידע הרלוונטי. רכיב המנהל מחזיר אובייקט המכונה `FSDataInputStream` המכיל את כל המידע על רכיבי הפועלים הדרוש, בדומה לאובייקט המוחזר בפעולת הכתיבה.
3. רכיב הלקוח פונה אל האובייקט שקיבל ומתחיל לקרוא את הבלוק הרלוונטי. האובייקט דואג לשלוח את הבלוק מרכיב הפועל המתאים.
4. המידע מועבר אל רכיב הלקוח.
5. אובייקט ה `FSDataInputStream` מכווון אל רכיב הפועל הבא ומזרים את המידע אל רכיב הלקוח. בעת הצורך, פונה שוב אל רכיב המנהל כדי לדעת מהיכן לשלוח את הבלוק. הפעולה שקופה לרכיב הלקוח, הלקוח מבצע קריאה חוזרת ונישנה מהאובייקט.
6. לאחר סיום פעולת הקריאה, האישור מדווח אל רכיב המנהל, והפעולה מסתיימת.

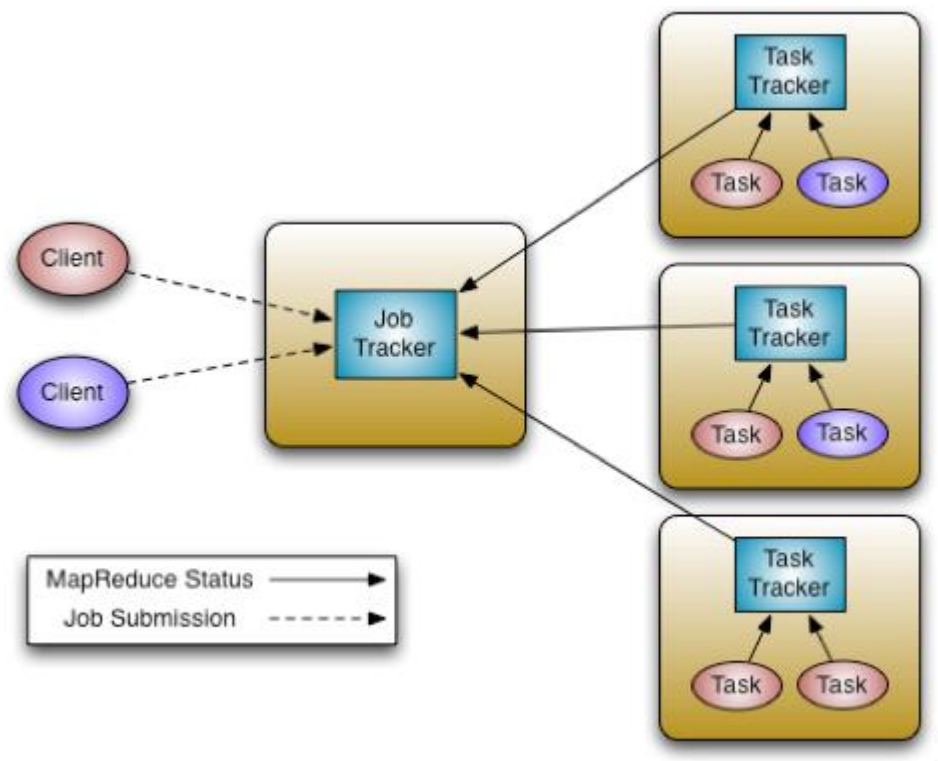
Hadoop MapReduce 3.5.3

Hadoop MapReduce הוא קוד פתוח שפותח על ידי אפאצ'י בשפת Java. המודל עובד על עיקרון של גוגל MapReduce (סעיף 3.4).

באיור 19 ניתן לראות את MapReduce בגרסה 1.0 של Hadoop

התקנה של Hadoop, מורכבת מצומת ראשית אחת וצמתי פועלים רבים. הצומת הראשית מריצה תהליך **JobTracker** האחראי על ניהול מחזור חיים של ה job (תזמון משימות, מעקב התקדמות, עמידות מבפני תקלות), ניהול משאבים (לדוגמא ניהול את ה TaskTrackers), ומעקב אחר צריכת/זמינות משאבים. JobTracker מתקשר עם NameNode, כדי לאתר את המיקום המידע. כל פועל מריץ תהליך **TaskTracker** המשגר משימות לפי הסדר ש JobTracker מכתוב, ומדווח על מצבן ל JobTracker. כל TaskTracker מוגדר עם סט של חריצים, המצביעים על מספר המשימות שהוא יכול לקבל. משימה ב TaskTracker רצה בתהליך jvm נפרד (אם משימה נכשלת, TaskTracker ממשיך לעבוד). TaskTracker עקב אחר המשימות המתבצעות ומדווח ל JobTracker על הצלחה וכישלון של משימה. כל כמה דקות TaskTracker שולח הודעה עם מספר חריצים פתוחים ל JobTracker, כדי להודיע שהוא חי, ולעדכן את JobTracker עם כמות החריצים הפנויים.

רכיב הלקוח (Client), מעביר את העבודות (jobs) לריצה ל JobTracker, המחלק את העבודה למשימות (tasks) ומקצה את המשימות לצמתי פועלים לביצוע. [5].



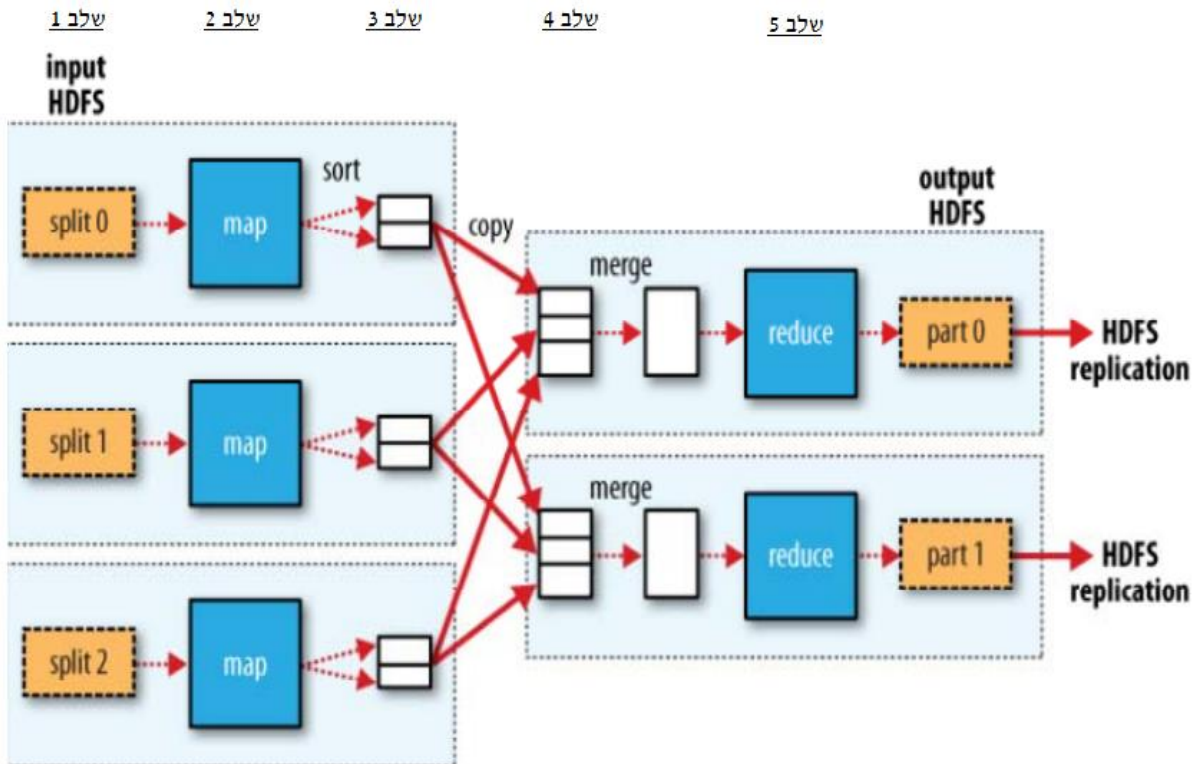
איור 19 – MapReduce בגרסה 1.0 של Hadoop [13]

3.5.3.1 שלבי זרימת מידע ב Hadoop MapReduce

קובץ קלט מחולק לבלוקים בדרך כלל בגודל 64 קילו ביט, עבור כל בלוק מורץ שלב המיפוי. כמות הבלוקים בקובץ קלט קובעת את מספר משימות המיפוי.

בהרצת משימת Hadoop, תוכנית משתמש יוצרת קובץ קונפיגורציה, המגדיר את הממשקים של שלב מיפוי, שלב צימצום ונתיבי קלט/פלט (ניתן להגדיר גם את מספר משימות הצמצום ופורמט הפלט). JobTracker שם את קובץ קונפיגורציה ותוכנית משתמש במיקום משותף, הנגיש לכל ה TaskTrackers. מעת לעת TaskTrackers מתשאלים את JobTracker אם קיימת משימה לביצוע, ובעת ביצוע המשימה שולפים את קובץ קונפיגורציה מהמיקום המשותף. כל משימה ב- TaskTracker רצה על jvm (מכונה וירטואלית המפרשת ומריצה java bytecode) אחר [3].

באיור 20, מודגם שלבי זרימת המידע ב Hadoop MapReduce.



איור 20- שלבי זרימת מידע ב Hadoop MapReduce [16]

- שלב 1 : המערכת מפצלת קלט (input HDFS) ממערכת הקבצים על פני צמתי מיפוי (map).
- שלב 2 : פונקציית מיפוי יוצרת פלט בצורת <מפתח, ערך>. לכל מפתח פונקציית מיפוי מחשבת את מספר המחיצה (מפעילה את פונקציית המחיצה על המפתח).
- שלב 3 : בכל צומת מיפוי, הנתונים ממויינים (sort), לפי מספר מחיצה ולאחר מכן לפי מספר מפתח. הנתונים נשמרים במערכת קבצים מקומית של צומת המיפוי (באיור 20 נוצרות שתי מחיצות)
- שלב 4 : המערכת מעבירה את המחיצות לשלב הצמצום. כל שלב צמצום מקבל את מרווח הנתונים (מפתח וערכים) ממחיצה אחת, לפי מספר המחיצה מכל שלבי המיפוי.
- לאחר מכן, מתבצע תהליך מיזוג (merge) וקיבוץ של הנתונים (הנתונים משלבי מיפוי ממויינים בכל מחיצה).
- שלב 5 : עבור כל מפתח, מופעלת פונקציית הצמצום של המשתמש (reduce) ורושמת את הפלט (part 0, part 1) במקום זמני ב HDFS

המערכת אוספת את כל הפלטים משלבי הצמצום, ויוצרת פלט סופי

שלבי ביצוע משימת מיפוי:

ביצוע משימת מיפוי מחולקת ל 2 חלקים :

1. שלב מיפוי:

- קורא בלוק מ HDFS, הופך אותו לרשומות (זוגות ערך, מפתח)
- מפעיל את פונקציית המיפוי של המשתמש לכל רשומה.

2. שלב הביצוע :

- ממזג את הפלטים של שלב המיפוי לפלט סופי
- TaskTracker מעדכן את JobTracker עם סיום המשימה.

באיור 21 ניתן לראות את הממשק למימוש של פונקציית מיפוי (map).

```
public interface Mapper<K1, V1, K2, V2> {  
    void map(K1 key, V1 value,  
            OutputCollector<K2, V2> output);  
  
    void close();  
}
```

איור 21 - ממשק פונקציית מיפוי [5]

לאחר מיפוי רשומות, מופעלת פונקציה close (משתמש מממש פונקציה זו, ויכול להגדיר בה, סגירות של אובייקטים וכו).

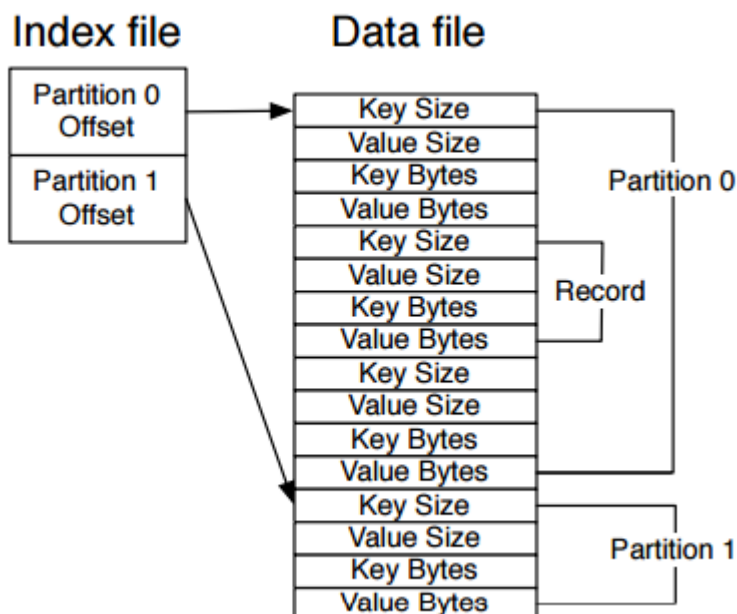
אחד הפרמטרים בפונקציית מיפוי הוא OutputCollector, מחלקה גנרית, הצוברת את רשומות הפלט. OutputCollector מפעיל את הפונקציה המחיצה (לפי המפתח) על כל רשומת פלט, ומאחסן את הרשומה ומספר מחיצה במאגר הזיכרון. פונקציית המחיצה מחושבת באופן הבא (בתוכנית משתמש ניתן ל"דרוך" על פונקציית המחיצה, ולממש במקומה פונקציה אחרת) :

```
partitionIdx = (key.hashCode() & Integer.MAX_VALUE) % numReducers
```

פרמטר numReducers (מספר שלבי הצמצום), נקבע על ידי המשתמש. סהכ נוצרות R מחיצות (מספר המחיצות שווה למספר שלבי הצמצום) .

גודל מאגר הזיכרון נקבע בקובץ קונפיגורציה (בדרך כלל 100 מגה ביט). כאשר מאגר הזיכרון מגיע לקיבולת מקסימלית, OutputCollector ממיינ את הרשומות לפי מספר מחיצה ומפתח, ומעביר את תוכן המאגר למערכת קבצים מקומית. הנתונים נשמרים בקובץ אינדקס וקובץ נתונים.

באיור 22, ניתן לראות את פורמט האינדקס וקובץ נתונים עבור שני מחיצות (numReducers=2).



איור 22 – פורמט אינדקס של משימת מיפוי, וקובץ נתונים [5]

קובץ אינדקס מצביע על כל מחיצה בקובץ נתונים. קובץ נתונים מכיל את הרשומות, הממויינות לפי מפתח בכל מגזר של מחיצה. במהלך שלב ביצוע, נוצר הפלט סופי של משימת המיפוי, כל נתונים ממוזגים לקובץ אחד של אינדקס ונתונים (הנתונים מחולקים ל R מחיצות, כאשר בכל מחיצה הנתונים ממויינים לפי מפתח).

שלבי ביצוע משימת צמצום:

משימת צמצום מתבצעת בשלושה שלבים:

1. **שלב העירבוב** – מביא את את הערכים משלב המיפוי בעזרת קריאות HTTP במקביל, למספר TaskTrackers בדרך כלל 5 (משימת צמצום מקבלת את הפלט של משימת מיפוי, רק לאחר שמשימת מיפוי, סיימה בהצלחה את הביצוע ורשמה את הפלט על הדיסק).

כל משימת צמצום מקבלת מחיצה אחת עם טווח המפתחות הנוצרים בשלב המיפוי מכל צמתי המיפוי (אם המשימה התחלקה ל M צמתי מיפוי, הנתונים יגיעו לכל היותר מ M צמתיים, רק מצמתיים בהם קיימים נתונים למפתח המחיצה).

2. **שלב המיון** – שלב זה מורכב מ**מיון** (הערכים במחיצות כבר ממויינים לפי מפתח) ו**קיבוץ** הערכים לפי מפתח (שלבי מיפוי שונים יכולים ליצור אותו מפתח).
3. **שלב הצמצום** - מפעיל את פונקציית הצמצום של המשתמש עבור כל מפתח (הפרמטרים לפונקציה, מפתח ורשימת הערכים של המפתח). הפלט של פונקציית צמצום נכתב למקום זמני ב HDFS. לאחר סיום כל הריצות של פונקציות הצמצום, המערכת יוצרת פלט סופי.

3.5.4 תזמון משימות

בגרסאות ראשונות של Hadoop, ברירת המחדל של תזמון משימות משתמש היה FIFO. בהמשך פייסבוק פיתחו מנגנון תזמון הוגן (Fair Scheduler) ו yahoo את מנגנון תזמון הקיבולת (Capacity Scheduler). **תזמון FIFO** – משימות משתמש רצות בהתאם לסדר הגעתן. התזמון לא תומך בעדיפויות. עבור כל משימה, Hadoop מקצה אשכול [17].

תזמון הוגן – הקצאת משאבים בין העבודות (jobs), באופן שווה, לאורך זמן. התזמון מאפשר קיבולת אשכול הוגנת לאורך זמן. במידה ומורצת עבודה אחת, היא מקבלת את כלל קיבולת האשכול עבורה. במידה ומשתמשים מריצים עבודות נוספות, חריצים עבור המשימות באשכול משותפים בין כל המשתמשים באופן הוגן. התזמון הוגן כלפי משימות ארוכות וקצרות באותה מידה, מאפשר לעבודות קצרות להסתיים בזמן סביר ללא הרעבת עבודות ארוכות.

התזמון יודע לתזמן עבודות עם עדיפויות על ידי אירגון העבודות ב pools (pool מקום לשמירת עבודות) והקצאת משאבים בצורה הוגנת בין ה pools. כל משתמש מקבל pool אחד, בו נשמרות העבודות שלו לביצוע. בתוך pool רץ תזמון הוגן או FIFO. לדוגמא, במערכת עם שני משתמשים ואשכול המחולק ל pools, אם כל אחד מהמשתמשים שולח 3 משימות לביצוע, כל משתמש, יקבל pool משלו באשכול עם שלושת העבודות, המתזמן יריץ את ששת העבודות במקביל.

במידה וה pool לא מקבל חלוקה הוגנת לאורך זמן, המתזמן רשאי להורג משימות ב pools בהן הקיבולת גבוהה, ולהריץ משימות נוספות ב pool עם הקיבולת נמוכה.

תזמון קיבולת – מקובל להתשמש בתזמון עבור סביבות מרובות משתמשים, בהן קיים צורך לבצע חלוקה הוגנת של המשאבים. האשכול בנוי מתורים, ולכל תור מוקצים משאבים. התזמון מתבצע בשיטת FIFO עם עדיפויות. הדבר מאפשר למשתמשים לדמות אשכול MapReduce נפרד עם תזמון FIFO.

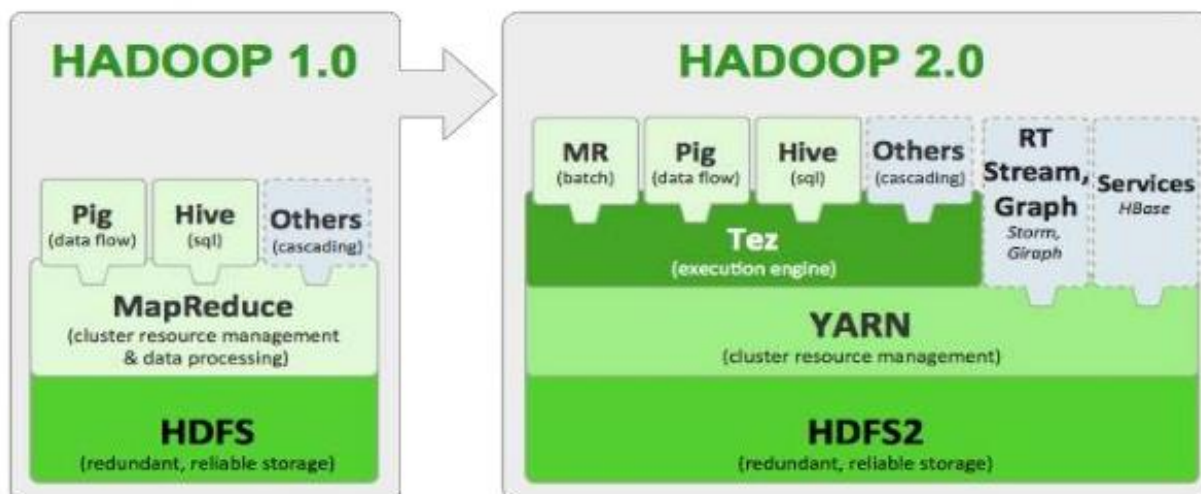
Job מריץ סט של משימות במקביל. חלק מהמשימות יכולות לרוץ זמן רב, ולעלות את זמן הביצוע הכולל של ה job. כדי להתגבר על הבעיה, Hadoop מזהה את המשימה האיטית ומריץ גיבוי נוסף של המשימה. הגיבוי רץ בזמן איטי יותר מהזמן הצפוי למשימה. אם המשימה המקורית מסתיימת לפני משימת הגיבוי, הורגים את משימת הגיבוי וההפך (בדרך זו מגבילים את הזמן של המשימה האיטית). השיטה יעילה במקרים בהם קיים עומס גבוה על מעבד, בעיות בחומרה וכד. השיטה לא יעילה במקרה והמשימה איטית בגלל באג במשימה, כי גם במשימת הגיבוי קיים הבאג (משימת גיבוי הוא העתק של המשימה המקורית).

YARN 3.5.5

עד לאחרונה, Hadoop היה מורכב בעיקר מ MapReduce. כדי להפוך אותו לג'נרי ולפתור את המגבלות של Hadoop, המחברים של Hadoop החליטו לנתק את ניהול המשאבים ממודל של ה MapReduce. מנהל המשאבים החדש הוא Yarn (Yet another resource negotiator) שהינו קוד פתוח, בגרסה 2.0 של Hadoop ומהווה מסגרת לניהול תהליכים ומשאבי אשכול, הפותרת את המגבלות העיקריות ב Hadoop [17]:

- יכולת ההרחבה של ה MapReduce (יכול להתרחב עד 4000 צמתים)
- חוסר יכולת שיתוף משאבים בין מסגרות רבות (לא רק MapReduce).
- ניצול האשכול. אשכול מורכב מצמתים עם חריצי map ו reduce – חריצים של map יכולים להיות מלאים בזמן שחריצים של reduce ריקים וההפך.

באיור 23 מודגמים השינויים שחלו בארכיטקטורת Hadoop בגרסאות 1.0 ו- 2.0.



איור 23 – ארכיטקטורת Hadoop בגרסאות 1.0 ו-2.0

הרעיון המרכזי של Yarn, לפצל שתי פונקציונליות עיקריות של JobTracker, ניהול ותזמון משאבים לשני דימונים שונים. כך, יהיה **ניהול משאבים** (ResourceManager) גלובלי אחד, וכן לכל יישום **אב יישום** (ApplicationMaster) אחר. אב יישום יקבל את רוב הפונקציונליות של JobTracker. JobTracker היה צוואר בקבוק, בגלל שקיים אחד באשכול, לעומת אב יישום הקיים לכל אפליקציה.

אב יישום מאפשר ל yarn להציע תכונות חדשות:

- הרחבה: אב יישום מספק את רוב הפונקציונליות של מנהל המשאבים המסורתי, לכן כל המערכת יכולה להתרחב באופן משמעותי. סימולציות הראו יכולת הרחבה ל 10,000 צמתים באשכולות, עם חומרה מודרנית וללא בעיות משמעותיות. הסיבה, לכל אפליקציה, קיים אב יישום מקומי משלו, המנהל את הקונטיינרים, עמידות מפני תקלות וכ"ד במקום מנהל משאבים גלובלי, אחד באשכול שגרם לצוואר בקבוק.
- העברת קוד של תשתית היישום לאב יישום, מאפשרת למערכת לתמוך במסגרות רבות ולא רק ב MapReduce.

הרכיבים של Yarn [13]:

- **מנהל משאבים** (ResourceManager) – מחלק משאבים בין יישומים. מורכב מ**מתזמן ומנהל יישום** (ApplicationManager).

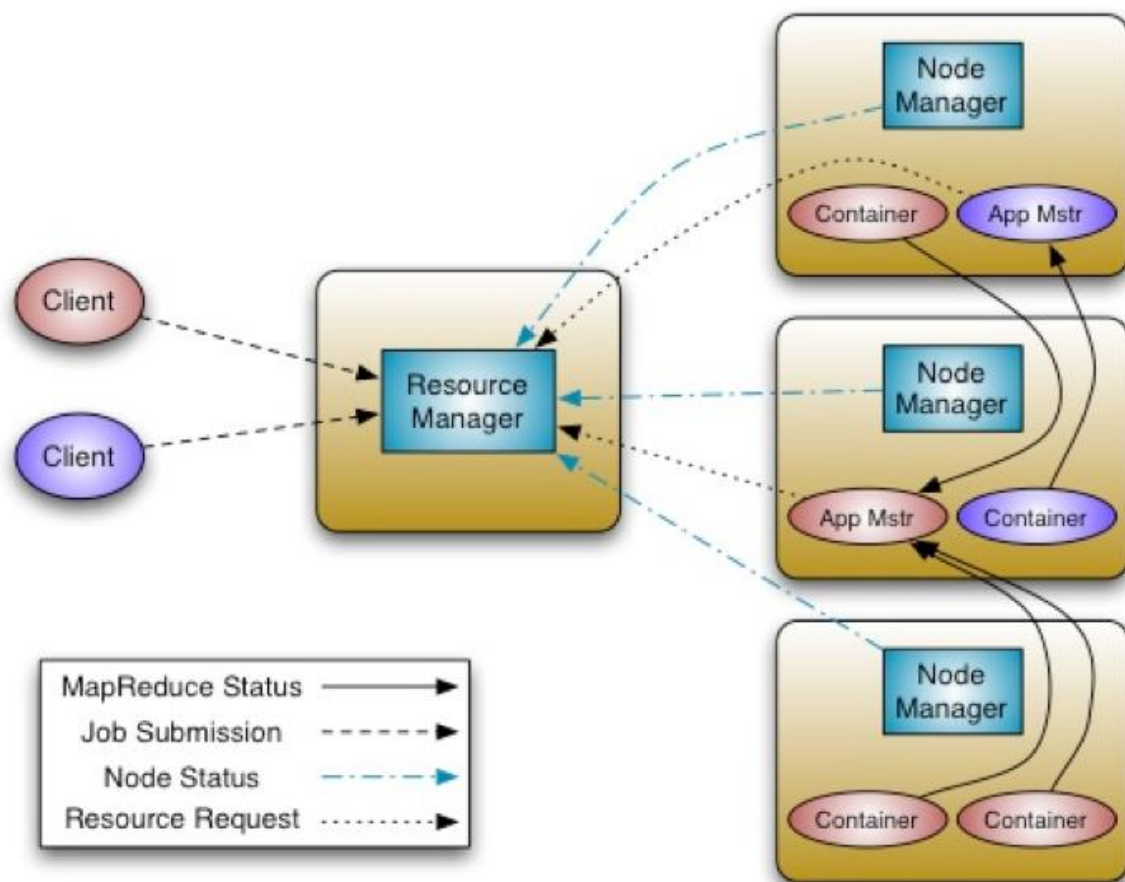
- **מתזמן**: מתזמן פשוט, המוגבל לתזמון משאבים בין היישומים. אלגוריתם התיזמון pluggable, ניתן להחליף אותו בכל אלגוריתם אחר לדוגמה תזמון הוגן או תזמון קיבולת.

התזמון מתבצע בהתאם לדרישת משאבים של היישום. המתזמן נחשב ל"פשוט" היות ומנהל רק את זמינות המשאבים ואוכף את מדיניות התזמון לפי ההגדרות קבצי תצורה (לא מטפל בניהול משימות, כגון ניטור או מעקב אחר סטטוס של יישומים. אם יישום נכשל, לא מאתחל אותו).

- **מנהל יישום (ApplicationManager)** : מקבל את העבודות לריצה, ומנהל את **אב יישום (ApplicationMaster)** : מטפל באב יישום הראשון שיריץ את המשימה, ומספק שירות לאיתחול מחדש של אב יישום בעת כישלון.

- **אב יישום (ApplicationMaster)** - לכל יישום יש אב יישום פרטי שלו הנחשב לקונטיינר. אב יישום אחראי על:
 - עבודה עם מנהלי צומת, לביצוע וניטור קונטיינרים וצריכת משאבים שלהם (מצב של קונטיינר, התקדמות, שיגור, טיפול בכישלונות).
 - ניהול משא ומתן עם מנהל משאבים לקבלת משאבי מחשוב.
- **קונטיינר (Container)** - מכיל את המשימות לביצוע (מריץ את ה task) והקצאה פיזית של משאבים (מעבד, זיכרון, דיסק, רשת).
- **מנהל צומת (Node Manager)** - מנטר את הצמתים ומנהל את מחזור החיים של קונטיינרים. מקבל הוראות ממנהל משאבים, מנהל את המשאבים של הקונטיינר (מעבד, רשת, זכרון, דיסק) ומדווח את הסטטוס לאב יישום.

באיר 24 מודגם התהליך של Yarn בגרסה 2.0 של Hadoop :



איור 24 – Yarn בגרסה 2.0 של Hadoop [31]

להלן תאור התהליך [34]:

- רכיב לקוח (Client) שולח עבודה (job) לביצוע, כולל כל התוכן הדרוש (דרישת משאבים, קבצים, אסימוני אבטחה וכ"ו) למנהל משאבים
- מנהל משאבים (Resource Manager) מבקש ממנהל צמתים (Node Manager), לאתחל את אב היישום (Application Master).
- מנהל צמתים מאתחל את אב היישום. אב היישום נרשם אצל מנהל המשאבים.
- מנהל משאבים משתף יכולת משאבים עם אב היישום.
- אב יישום מבקש קונטיינרים (Containers מכילים משימות ומשאבים) ממנהל משאבים.
- מנהל משאבים מקצה קונטיינרים בהתאם למדיניות וזמינות המשאבים.
- אב יישום מתקשר עם מנהלי הצמתים, ומבקש מהם לאתחל את הקונטיינרים.
- מנהלי צמתים מאתחלים את הקונטיינרים (המשימות מתחילות לרוץ), והעבודה מתחילה להתבצע.

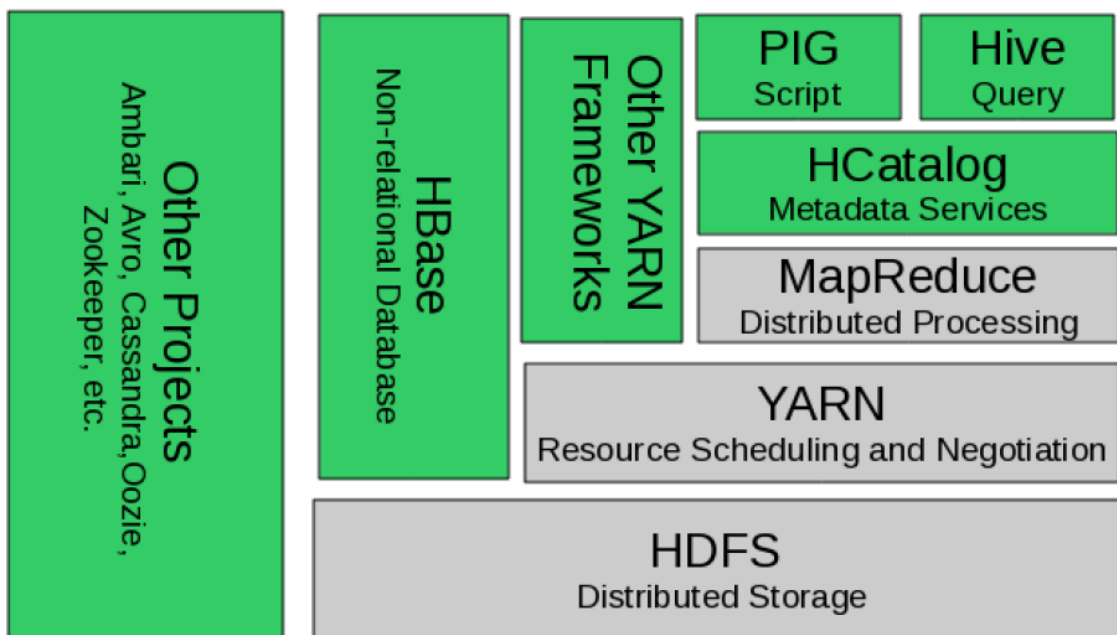
במהלך הריצה :

- אב יישום מבקש את הסטטוס של הקונטיינרים ממנהלי צמתים
- אב יישום מבקש הקצאה/שחרור של קונטיינרים ממנהל משאבים
- בהקצאה חדשה, אב יישום מתקשר עם מנהלי צמתים לאיתחול הקונטיינרים
- שחרור קונטיינרים עובר דרך מנהל משאבים

מערכת האקו של Hadoop בגרסה 2.0

בגרסאות קודמות, JobTracker היה אחראי על ניהול משאבים ויישום והריץ רק משימות של Hadoop MapReduce (עבור כל עבודה ב Hadoop, היו חייבים להשתמש ב MapReduce). מנהל המשאבים החדש (Resource Manager) מאפשר להריץ שירותים אחרים באותו אשכול.

באיר 25 ניתן לראות את מערכת האקו של Hadoop בגרסה 2.0. ניתן לראות שהוסף ממשק כללי הממקם את MapReduce כאחת ממסגרות היישום ב Hadoop ומאפשר ל Hadoop להריץ מסגרות נוספות שאינן MapReduce (תשתית ה MapReduce נשארה ללא הרבה שינויים, כדי לתמוך בגרסאות ישנות). תשתיות הרצים ב Yarn, מתאמים התקשרות בין התוכניות, זרימת ביצוע, אופטימיזציות דינמיות ושיפורי ביצועים. Yarn כברירת מחדל משתמש בתזמון קיבולת.



איור 25 – מערכת אקו של Hadoop בגרסה 2.0 [13]

HBASE 3.5.6

Hadoop מתאימה לעיבודי אצווה. על מנת, לעבוד בזמן אמת עם Hadoop ניתן להשתמש ב HBase [27]. HBase בסיס נתונים NoSql (שכבת אחסון נתונים מבוזרת של רשומות מפתח וערך) בשפת Java, מבוסס Google Big Table וממומש מעל HDFS. HBase מאפשר גישה רנדומלית לקריאה/כתיבה ושימוש מלא ב-MapReduce (באמצעות API). HBase תומך בעמודות דינמיות (אין צורך להגדיר מראש את רשימת העמודות בכל טבלה) ומאפשר טעינה ואחזור מהיר לפי המפתח הראשי של הטבלה. לדוגמא HBase משמש כתשתית ההודעות של פייסבוק, לאחסון הודעות והציאת של האתר.

בהשוואה למנועי NoSQL, HBase היא הבחירה הטובה ביותר לעבוד עם Hadoop. במנועי NoSQL מרכזיים כגון קסנדרה יש צורך להתאים את התשתית כדי לעבוד עם HDFS (בקסנדרה כל הצמתים שווים לעומת HDFS בו קיימת צומת ראשית המנהלת את שאר הצמתים). ל MongoDB אין צורך לעבוד מעל Hadoop, קיים לו פיתרון משלו ליכולת הרחבה אופקית ועבודה עם נתונים המאוחסנים על פני מספר מחשבים.

3.5.7 השוואה בין Hadoop ל- RDBMS

רוב מערכות הבראות משתמשות כיום ב RDBMS. מערכות אלו, לא מתאימות לאחסון מסיבי וניתוח יעיל של נתונים. Hadoop מספק פיתרון לבעיה זו, והוא בעל עם תשתית אמינה וביצועים טובים [20]. בטבלה 3, מוצג ניתוח השוואתי בין Hadoop ל- RDBMS [32].

מדדי השוואה	Hadoop	RDBMS
תשתית	מבוסס צומת בעל מבנה שטוח, תשתית מערכת אקו הבנויה מפרוייקטים של Java	מערכת לניהול בסיסי נתונים יחסיים, פרויקט אחד בעל מספר רכיבים
ארכיטקטורה	מבוסס על מערכת קבצים כגון HDFS. תוכנן לתמוך בארכיטקטורה מבוזרת	מסתמך על מערכת קבצים של מעה"פ. תוכנן לתמוך בארכיטקטורת שרת לקוח
מחיר	קוד פתוח	בעיקר קנייני
שימוש עיקרי	משמש לניתוחים אנליטיים בעיקר עבור נתונים בקנה מידה גדול	משמש בעיקר לעיבוד תנועות מקוונות
הוספת כח עיבוד	להוספת כח עיבוד יש להוסיף צמתים חדשים	כדי להוסיף כח עיבוד כגון מעבד, זיכרון בסביבה לא וירטואלית יש צורך בהשבתת RDBMS
איחסון נתונים	בשלב של התפתחות	ניתן לאחסון נתונים ללא תלות בכל צומת חישוב
פופולריות	עדיין מתפתח	קיימים מוצרים טובים בשוק כגון אורקל, SQL
סכימה	תמיכה טובה גם בנתונים לא מובניים	חייב נתונים מובניים
כתיבה רציפה	תומך בכתיבה רציפה	תומך ביצירה ועידכון שרירותים

טבלה 3 : ניתוח השוואתי בין Hadoop ל RDBMS

4 סקר ספרות אודות יישומי מחשוב מובייל ענן

4.1 מבוא

כיום קיימות מערכות מובייל ענן רבות, המשתמשות בכח העיבוד והאיחסון של ענן דרך המובייל. נציג את תחומי היישום העיקריים [6], [2] :

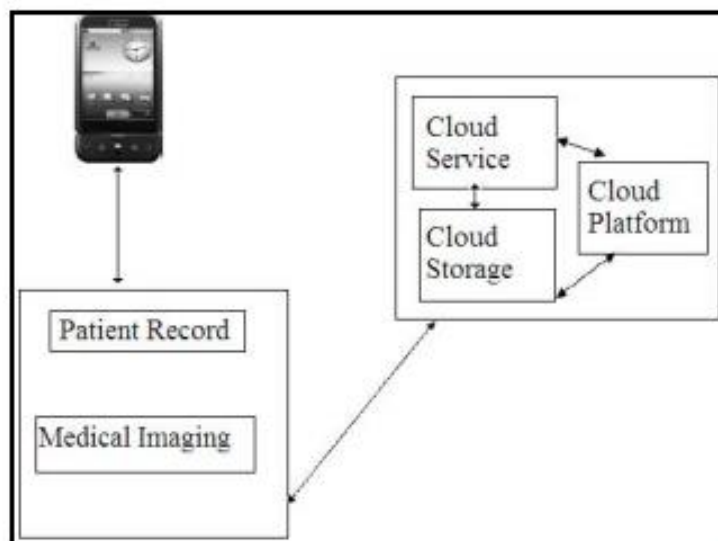
- מסחר - מודל עסקי למסחר באמצעות מכשירים ניידים. פרסומות, קניות, נושאים פיננסיים (המערכות נמצאות בענן, דרך המובייל מבצעים את הפעולות).
- למידה אלקטרונית/ משחקים - לימודים אלקטרוניים/משחקים דרך המובייל (התוכנות, המשחקים והמנוע נמצאים בענן, דרך המובייל מפעילים את היישומים).
- בנקאות - מתייחסת לכל פעולה הקשורה לשירות בנקאי כגון בדיקת יתרה, תשלומים, sms בנקאי באמצעות מובייל
- מערכות ממשלתיות – קיימים שירותי ממשל שניתן לנהל אותם דרך המובייל בשל הזמינות המיידית
- מערכות רפואיות – מאפשרות גישה לנתונים מכל מקום על ידי המובייל, דוגמא למערכות רפואיות:
 - שירותי ניטור רפואיים המחוברים לטלפון הנייד על ידי תקשורת אלחוטית ומאפשרים לחולים להיות במעקב בכל רגע נתון ומכל מקום
 - מערכת לניהול אמבולנסים, המתאמת ומנהלת אמבולנסים למיקום התאונה
 - מערכת המנטרת דופק, לחץ דם, רמת אלכהול ורמת סוכר ומתריעה בעת מצב חירום (מכשיר המחובר למובייל, ומתריע דרכו לענן בו יושבת המערכת).
 - מערכת לניהול תמונות רפואיות במובייל ענן - המערכת מפורטת בסעיף 4.2.
 - מערכת לניטור חולים בעלי אי ספיקה לבבית – המערכת מפורטת בסעיף 4.3.
 - מערכת לניטור וניתוח נתוני אק"ג בזמן אמת בענן- המערכת מפורטת 4.4.

4.2 מערכת לניהול תמונות רפואיות במובייל ענן

המערכת מנהלת תמונות רפואיות ונתוני מטופלים בענן [23], ומציגה את הנתונים (נתונים אישיים, מרשמים ותמונות רפואיות) בטלפון חכם (כח המחשוב והאיחסון נמצאים בענן). מערכת זו מהווה מערכת טלמדיסין רפואה מרחוק, המערכת חוסכת מהמטופל להגיע למרפאה (המטופל יכול לראות את כל הנתונים דרך הטלפון החכם), ומאפשרת גישה לנתונים מכל מקום למטופלים ולצוות הרפואי [23].

במערכת היא מערכת שרת לקוח :

- השרת בענן . בשרת מותקן בסיס נתונים רציונלי אורקל, במטרה לתמוך ב ACID ולהחזיר נתונים מדויקים ללקוחות.
 - הלקוח, הוא מובייל כולל ממשק וובי.
 - מובייל (רק עבור מערכות הפעלה אנדרואיד) משרת את המטופלים. ממשק בטלפון חכם, מאפשר למטופלים לראות את הנתונים הפרטיים ותמונות רפואיות.
 - ממשק וובי, משרת את הנהלת בית החולים ורופאים. הממשק מאפשר לרופאים לקבוע ביקורי מטופלים, לעדכן מרשמים של המטופלים ולהעלות תמונות רפואיות לענן.
- התקשורת בין הלקוח לשרת מתבצעת בעזרת שירותי אנטרנט עם פרוטוקול Rest.
- התמונות נדחסות בהתאם לסטנדרטיים של JPEG ומועברות בעזרת פרוטוקול DICOM. במערכת הפעלה אנדרואיד (אנדרואיד SDK, סט כלים לפיתוח תוכנה) ישנן תכונות המאפשרות לתוכן שהתקבל, להיות מותאם לגודל המסך.
- באיור 26 מוצגת ארכיטקטורת המערכת לניהול תמונות רפואיות.



איור 26 – ארכיטקטורת המערכת לניהול תמונות רפואיות [23]

המטופל מזדהה מול השרת בענן על ידי שם משתמש וסיסמא, כאשר שם המשתמש של המטופל הוא תז. בכניסה למערכת, כל רשומות המטופל נשלפות מידיית מהענן (לפי תז שהוא המפתח הייחודי) ומוצגות בטלפון החכם. כאשר מטופל מעוניין לראות תמונות רפואיות, נשלחת בקשה לענן באמצעות פרוטוקול Rest הדומה מאוד לבקשת HTTP URL. מערכת ההפעלה בענן בודקת את הבקשה בבסיס נתונים ומחזירה תשובה לטלפון החכם. בטלפון החכם, היישומים הניידים רצים כל הזמן ובודקים האם הגיע קובץ וממתין להורדה, אם כן מורידים את הקובץ ומציגים אותו למטופל.

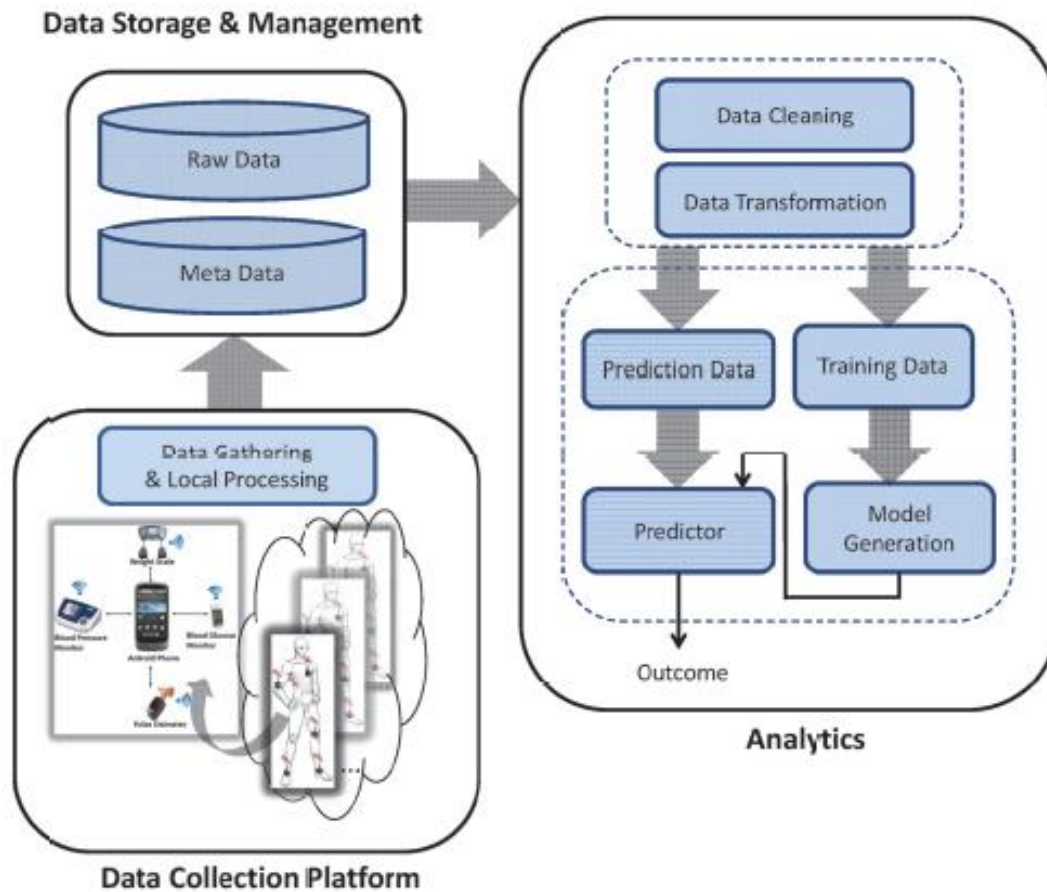
עבודה עתידית במערכת זו, הוא יישום נושא הסודיות והפרטיות בנתונים רפואיים.

4.3 מערכת Wanda לניטור חולים בעלי אי ספיקה לבבית

מערכת Wanda היא מערכת לניטור וניתוח נתונים של חולים בעלי אי ספיקה לבבית [10]. המערכת מורכבת מאיסוף מידע בעזרת טלפון חכם, איחסון המידע הניתן להרחבה באינטרנט ומנוע אנליטי למטרות אבחון ופרוגנוזה. המערכת תומכת באיסוף נתונים ממגוון רחב של חיישנים המודדים נתוני נבדק בדידים ורציפים כגון משקל, קצב לב, רמת סוכר בדם, רמת חמצן, לחץ דם ושאלוני דיווח עצמיים. המטרה העיקרית של המנוע, לחזות אירועים עתידיים של התקפי לב וזאת על ידי בדיקת נתוני בדיקות שנאספו על החולים. המערכת רצה רק על מכשירים עם מעה"פ אנדרואיד החל מגרסה 2.0 ומעלה. בהמשך נסקור בהרחבה את מבנה המערכת.

ארכיטקטורה:

ארכיטקטורת המערכת מוצגת באיור 27.



איור 27- ארכיטקטורת מערכת Wanda [10]

המערכת בנויה משלוש שכבות :

- השכבה הראשונה **Data Collection Platform** - תשתית לאיסוף נתונים המורכבת מאוסף חיישנים המודדים נתונים שונים בגוף. הנתונים מועברים דרך היציאה של הטלפון החכם לענן.
- השכבה השנייה **Data Storage & Management** - ענן. הנתונים נשמרים בענן בבסיס נתונים NoSql עם יכולת הרחבה ואינדוקס. הגישה לנתונים מתבצע דרך ממשק וובי RestfullStyle.
- השיכבה השלישית **Analytics** - מנוע אנליטי, הכולל אלגוריתמים של כריית מידע כדוגמת אלגוריתמי אשכול וכ"ו המשמשים למטרות איבחון וחיזוי. לדוגמא, עבור חולים בעלי אי ספיקה לבבית, ניתן לחזות החרפה בתסמינים טרם אישפוז החולה. כך, ניתן להיערך מבעוד מועד עם העירכות מוקדמת של צוות רפואי למתן טיפול יעיל ומהיר.

ממשק משתמש

למערכת קיים ממשק לטלפון חכם (עם מערכת הפעלה אנדרואיד). הממשק מאפשר ללקוח לבחור את סוג המדידה המתאים, לבצע את המדידה המבוקשת, ולהציג באופן גרפי את תוצאות המדידה שבוצעו. כאשר לקוח רוצה לקחת מדידה, צריך ללחוץ על כפתור מתאים ליצירת התקשרות עם בלוטוס. המדידה נשמרת במכשיר, מועברת כחבילה אחת לטלפון החכם ומציגה את המדידה למשתמש. ניתן לראות את הממשק באיור 28.



(a)



(b)



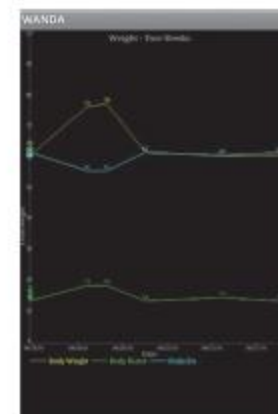
(c)



(d)



(e)



(f)

איור 28 – ממשק אנדרואיד במערכת Wanda [10]

- (a) מסך ראשי: משתמש יכול לבחור להקליט מדידה חדשה או לצפות בהיסטוריית המדידות שלו.
- (b) הקלטת מדידה חדשה: מוצגות ארבע אפשרויות מדידה: משקל, לחץ דם, סוכר בדם, וקול.
- (c) בעת לחיצה על אחת המדידות (ובהתאם בחירת מכשיר רפואי), מוצגים למשתמש הנחיות למדידה. להתחלת ביצוע המדידה על המשתמש ללחוץ על כפתור "Start Measurement" וזאת על מנת לבצע התקשרות בבלוטוס עם מכשיר הרפואי ולהתחיל במדידה.
- (d) בסיום המדידה, הטלפון החכם מקבל את המדידה דרך הבלוטוס כחבילה אחת, ומציג את נתוני המדידה למשתמש. המערכת חוזרת לאיור b, על מנת לאפשר למשתמש לבצע מדידה נוספת.
- (e) בבחירת "היסטוריית מדידות" (במסך ראשי), ללקוח מוצג דף הדומה לאיור b, כדי לאפשר למשתמש לבחור את מכשיר המדידה. לאחר בחירת המכשיר, באיור e מוצג למשתמש, מסך עם פרקי זמן. המשתמש בוחר פרק הזמן כדי לראות את היסטוריית המדידות של המכשיר.
- (f) היסטוריית מדידות של המכשיר מוצגות בגרף.

שלב התקשרות

הממשק לטלפון החכם מתבצע על ידי בלוטוס. טרם יצירת קשר בין המכשיר לטלפון החכם, יש לבצע תהליך של זיהוי. כאשר שני מכשירי בלוטוס נמצאים אחד ליד השני, קיים להם מפתח קישור לזיהוי בעזרתו יכולים ליצור התקשרות מוצפנת. לאחר הזיהוי, פרטים של המכשיר מועברים לטלפון החכם, ונשמרים במערכת פעם אחת. פרטי המכשיר המועברים הם: שם המכשיר, Mac Address של המכשיר וסוג המכשיר.

שלב ביצוע המדידה

כל מדידה מתבצעת בעזרת מכשיר מתאים, ובסיום מועברת כיחידה אחת לטלפון החכם (גם בנתונים רציפים, המדידה נשמרת במכשיר ובסיום מועברת כיחידה אחת לטלפון החכם). המערכת מאחסנת את הנתונים בענן, מנתחת את הנתונים על ידי מנוע איבחון ומתריעה על אירועים חריגים בעזרת בקשות HTTP בממשק או על ידי שליחת מייל.

המערכת משתמשת בענן S3 של אמזון. אמזון מספקת אבטחה באמצעות מנגנון לבקרת גישה מרובה והצפנה של נתונים בהעברה ובדיסק.

אמינות, עמידות, אימות ותיקון של איחסון פגום מובטחים על ידי אמזון. במערכת משתמשים ב SimpleDB של אמזון, כדי לאחסן נתונים מובניים. SimpleDB הוא בסיס נתונים NoSQL המציע הרחבה ללא הגבלה וסכמה גמישה לנתונים. הגישה לנתונים ב NoSQL בעייתית, חסרות לו הרבה מהתכונות הקיימות בבסיס נתונים רלציוני כגון פעולות צירוף וערבות מלאה ל ACID (לא ניתן לגשת לבסיס נתונים על ידי שאילתות SQL). במערכת כזו קשה לפתח תוכנה המתקשרת ישירות עם בסיס נתונים, ולכן פיתחו במערכת שיכבה נוספת (שכבת ביניים בין בסיס נתונים ללוגיקה עיסקית של המערכת) עם ממשק RESTful וובי המתקשרת עם בסיס נתונים על ידי אובייקטים בפסאודו. ניתן לראות דוגמא בטבלה 3

Resource	Methods	Result
/measurements	GET	Return a list of latest measurements
/subjects/add	PUT	Add a new subject
/sensors/<Sensor ID>	GET PUT DELETE	Return info of a specific sensor Update info of a specific sensor Remove a specific sensor

טבלה 3: משאבי RESTful במערכת Wanda

כדי לקבל את המדידות של המכשיר, משתמשים במתודה Get, להוספת מדידה בפונקציה Put וכ"ד.

בדיקת נכונות

במשך שלושה חודשים, התבצעה בדיקת מדידה ב 1500 מטופלים עם בעיות של אי ספיקה לבבית, בגילאי 50 ומעלה. המטופלים על בסיס יומי, מדדו משקל, לחץ דם, סוכר וענו על שאלון דיווח עצמי. הנתונים נשמרו בענן (עברו לענן דרך הרשת הסוללרית). המנוע האנליטי בנה מודלים לחיזוי עם דיוק של 74 אחוז. מערכת Wanda ידעה לנבא במדויק החרפת תסמינים בחולים בעלי אי ספיקה לבבית.

יתרונות : הנתונים במערכת נשמרים בבסיס נתונים NoSQL הניתן להרחבה עם ביצועים מהירים. נבנתה

שכבה נוספת המאפשרת לתקשר עם בסיס נתונים באמצעות אובייקטים (ממשק Restfull).

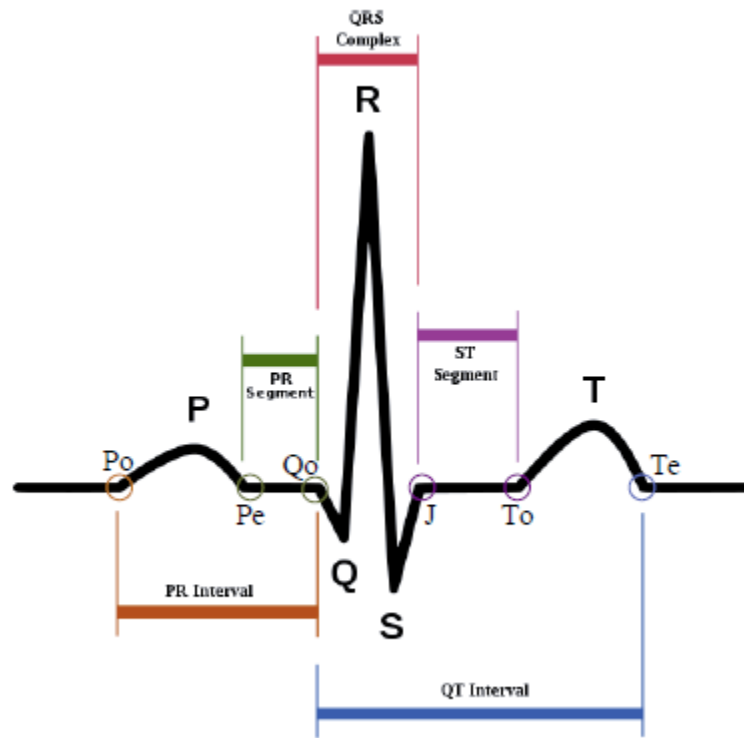
חסרונות : המערכת לא מבצעת ניתוח בזמן אמת. לא ניתן להשתמש בשפת SQL לשליפת שאילתות.

(במערכת, בסיס נתונים משמש רק עבור ניתוח אנליטי, לכן אין צורך בתמיכה ב ACID).

4.4 מערכת לכריית נתוני א"קג בזמן אמת בענן

פותחה מערכת לחיזוי מוקדם של בעיות לבביות על סמך נתוני א.ק.ג של נבדק המשודרים באמצעות רשת אלחוטית והמאוחסנים בענן [28].

מבנה אקג הבסיסי עבור פעימת לב אחת



איור 29 – תרשים אק"ג עבור מחזור אחד של פעילות לבבית [28]

פעילות לבבית מורכבת מפעילות מכנית וחשמלית, שתי פעילויות הולכות ביחד. שריר לב מתכווץ ברגע שמגיע אליו אות חשמלי ונרפה ברגע שאות זה נפסק. פעילות חשמלית בא לידי ביטוי בגלים. תבנית אק"ג רגילה מורכבת מכמה גלים ופסגות המתרחשים במהלך מחזור הלב. באיור 29 ניתן לראות מחזור אחד של אות אקג. אות אקג מאופיין בחמש פסגות ועמקים המסומנים באמצעות אותיות P, Q, R, S, T. ניתוח אקג תלוי בזיהוי P, Q, R, S, T. גל P מייצג הפעלה של התאים עליונים של הלב והפרוזדורים, מכלול QRS וגל T מייצגים את העירור של חדרי הלב או את התאים התחתונים של הלב. זיהוי מכלול QRS היא המשימה החשובה ביותר בניתוח אותות אקג. ברגע ש-QRS זוהה, בדיקה מפורטת יותר של אות אקג, הכוללת קצב הלב, מקטע ST וכ"ו יכולה להתבצע.

כל מרווח QRS (מחזור אחד של פעילות לבבית) מורכב מגלים וסיגמנטים רבים, אזורים מסויימים במרווח QRS חשובים יותר מאחרים. המאפיין הצורני של מרווח QRS משתנה עבור אנשים שונים, ועבור אותו בן אדם במצבים שונים. בנוסף מרווח QRS מתאים לקצב פעילות הלב, הגלים יכולים להיות באורכים שונים. בהתחשב במאפיינים אלו, מגדירים מדד, המודד את השוני בין שני מרווחי QRS. עבור שני מרווחי QRS, Q ו R השוני מסומן על ידי $Dis(Q;R)$ (ניתן לראות במאמר [28] איך מחשבים את השוני בין שני מרווחי QRS).

כריית מידע ואשכול

כריית מידע זהו תהליך ניתוח אוטומטי של מאגרי נתונים גדולים או מורכבים, על מנת לחשוף תבניות או מגמות, אותן לא ניתן היה לגלות בדרך אחרת. אחת הגישות לכריית מידע הוא ניתוח אשכולות. אשכול היא קבוצת רשומות. המטרה באשכול, לקבץ אובייקטים לקבוצות (אשכולות) כך שהאובייקטים הנמצאים באותה קבוצה דומים זה לזה יותר מאשר לאובייקטים השייכים לקבוצות אחרות. האלגוריתם הבסיסי לניתוח אשכולות הוא k-means. אלגוריתם בוחר k נקודות אקראיות (הנקודות הם אובייקטים, k נבחר על ידי משתמש) שהם המרכזים ההתחלתיים של האשכולות. כל נקודה שייכת למרכז הקרוב אליה ביותר (לפי מרחק האוקלידי של כל הנקודות מהמרכזים) בצורה זו מקבלים K אשכולות זרים זה לזה (בהמשך קובעים נקודת מרכז חדשה ע"י חישוב ממוצע של כל נקודות באשכול, אם נקודת מרכז שווה לנקודה קודמת התהליך מסתיים). בעזרת אשכולות, ניתן לקבל סקירה של נתונים, וזיהוי חריגים. ניתוח אשכולות מבוצע לעיתים כתהליך מקדים להפעלת משימות אחרות של כריית מידע.

המערכת

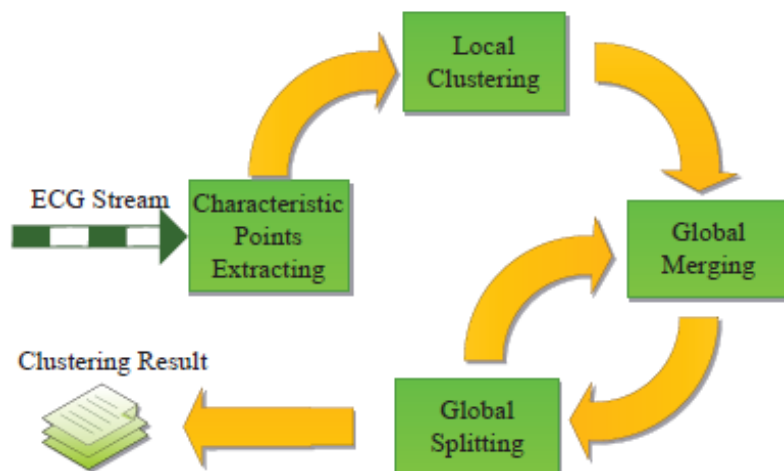
נתון שידור של רצף נתוני אקג לנבדק מסויים, נתונים אלו מאוחסנים בענן ובו זמנית עוברים הליך של כריית מידע בגישה של אשכול (הכרייה מתבצעת בענן). הליך כרייה מבוסס אלגוריתם k-means, המפעל את הנתונים לאוסף של אשכולות. ההליך החישובי מבוצע מעל ה MapReduce, החישובים לבניית האשכולות, מתבצעים במקביל על מספר רב של צמתים. תוצרי האשכולות נשמרים במערכת קבצים של HDFS, Hadoop.

אלגוריתם לבניית אשכולות מרצף של נתונים

אלגוריתם לבניית אשכולות מרצף של נתונים מציב דרישות חדשות (לעומת בניית אשכולות מנתונים קבועים). ראשית, הנתונים יגיעו ברציפות, כלומר האלגוריתם צריך להתבצע בזמן מוגבל. בנוסף, השינויים המתמשכים עלולים לגרום לאשכולות קיימים להשתנות (לעבור תהליך של מיזוג ופיצול). הרעיון של האלגוריתם, הנתונים החדשים שמגיעים מקובצים לאשכולות במחשב מקומי, מבלי לקחת בחשבון את הנתונים הקיימים (אשכולות קיימים המפוזרים במערכת קבצים של HDFS). הקשר והעיוות בין

האשכולות החדשים לאשכולות הקיימים בשרת, מוערך מחדש. אשכולות בהם קיים קשר גבוהה ימוזגו, אשכולות עם עיוות גבוהה יפוצלו לקטנים יותר. התהליך חוזר על עצמו בקבוצה הבאה של רצף נתוני אקג.

באיור 30 ניתן לראות את שלבי האלגוריתם.



איור 30 - אלגוריתם ליצירת אשכולות מתוך רצף של נתוני אקג לנבדק [28]

כאשר מגיעה קבוצה חדשה של רצף נתוני אקג, איטרציה המורכבת מארבעה שלבים מופעלת:

שלב 1 חלוקה של הזרם למרווחי QRS

מתבצעת חלוקה של נתונים למרווחי QRS בשרת מקומי.

שלב 2 יצירת אשכולות מקומית

מפצלים את המרווחים של QRS לקבוצות באופן רנדומלי (בשרת מקומי), לכל קבוצה מופעל אלגוריתם K-Means ליצירת k אשכולות, כאשר k משמש לבקרת מרכז הכובד של משקולות מקומיים. בהמשך, לכל אשכול מחושב ייצוג של אשכול (CF), באופן הבא:

$$CF(C) = (c, e, r, lr)$$

כאשר c הוא מרכז של אשכול C, e מרכז מרחב אוקלידי, lr הם ממוצע ומרחק מקסימאלי בהתאמה בין מרכז c למרווח QRS.

בשלב זה, טרם קיבלנו אשכולות אופטימליים (ייתכן ומרווחי QRS שצריכים להיות באותו אשכול, יהיו בקבוצות נפרדות), כדי לטפל בבעיה, מתבצעים שלבים של מיזוג ופיצול גלובלים של אשכולות (בניית

אשכולות התבססה על הנתונים החדשים, בסיום שלב 2, יתבצע תהליך של בניית אשכולות עם אשכולות מאיטרציה קודמת של נתונים, השמורים ב HDFS).

שלב 3 מיזוג גלובלי

המיזוג, מתבצע מול אשכולות קיימים ב HDFS.

מגדירים קשר בין שני אשכולות Q ו R -

$$con(Q, T) = \frac{(r_q + r_t)}{Dis(c_q, c_t)}$$

הפרמטרים r_q ו r_t הם רדיוס ממוצע של אשכול Q ו R, c_t ו c_q מרכזי אשכול Q ו R, $Dis(c_q, c_t)$ הוא מדד השוני בין c_t ל c_q . ממוזגים אשכולות המקומיים עם אשכולות מאיטרציה הקודמת. כל האשכולות שהקשר שלהם עונה על תנאי

$$con(Q, R) \geq \lambda$$

ממוזגים לאשכול אחד. הקשר צריך להיות גדול מפרמטר כלשהו שהוגדר מראש כדי לשלוט על מרכז הכובד של איחוד גלובלי. מיזוג גלובלי מתקן את הטעויות שנבעו מיצירת אשכולות מקומית ויכול לפגוע באיכות האשכול (בגלל מיזוג אשכולות הטרוגניות יחד). לאחר מיזוג גלובלי, איכות של כל אשכול מוערך מחדש.

איכות אשכול נמדדת לפי שגיאת עיוות. שגיאת עיוות מוגדרת באופן הבא :

$$DST(C) = \frac{w_1 * r + w_2 * Dis(c, e)}{lr}$$

$DST(C)$ שגיאת עיוות של אשכול C, w_1 ו w_2 משקלות יוריסטיים (דרך לחישוב המשקולות ניתן לראות במאמר [28]), r מרחק ממוצע של אשכול C, c הוא מרכז אשכול C, e הוא מרכז מרחב אוקלידי, Dis מדד שוני בין c ל e .

שלב 4 פיצול גלובלי :

לאחר הערכת אשכולות, על כל האשכול העונה לתנאי

$$DST(C) \geq \tau$$

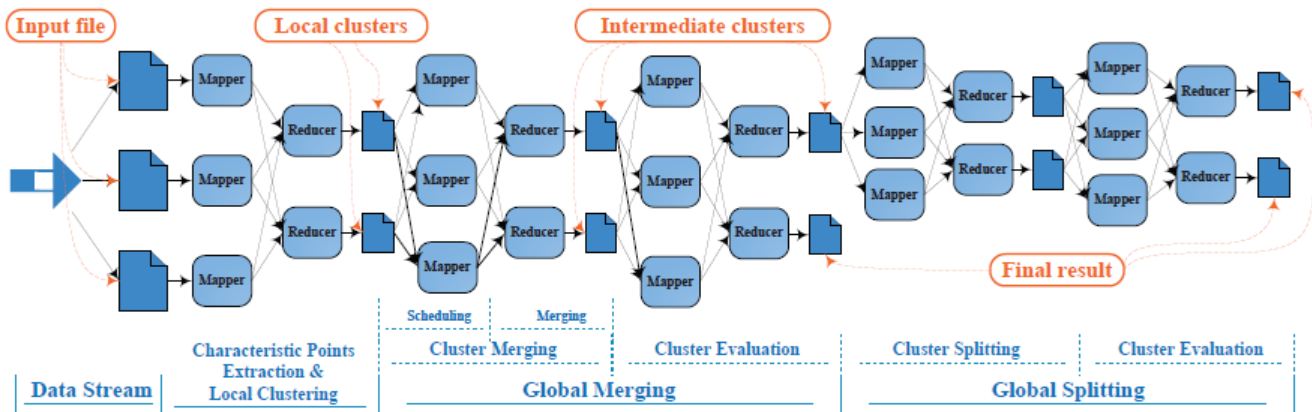
(שגיאת עיוות צריכה להיות גדולה יותר מפרמטר שהוגדר מראש עבור בקרת מרכז כובד של פיצול גלובלי) יופעל אלגוריתם K-Means לפיצול האשכול, לשני אשכולות קטנים יותר. פיצול מתבצע בשרת מקומי (לאחר המיזוג, האשכולות מעודכנים בשרת מקומי).

בסיום כל השלבים, האשכולות נשמרים ב HDFS, ומתחילה איטרציה על הקבוצה הבא של נתונים.

ריצה מקבילית של שלבי האלגוריתם

המערכת מעבדת זרם של נתונים המסודרים בקצב שידור גבוהה. היקף נתונים גדל אקספוננציאלית כתלות בזמן. לכן, המערכת יושמה מעל תשתית MapReduce, המאפשרת לטפל בנתונים מסיביים ולעמוד באתגר של מיזוג ופיצול אשכולות בזמן מוגבל. בפיתוח המערכת ניתן דגש רב על חלוקת שלבי האלגוריתם המודגם באיור 30, לחלקים בלתי תלויים, כך שעל כל חלק ניתן יהיה להפעיל מקביליות (בכל חלק, את החלקים התלויים מריצים סדרתית ואת השאר במקביל באמצעות פונקציות מיפוי וצמצום).

כל ארבעת השלבים של האלגוריתם, מפוצלים ל 5 שלבים ב MapReduce. איור 31 מציג את החלוקה



איור 31 – זרימת המערכת בהיבט של MapReduce [28]

- שני השלבים הראשונים של האלגוריתם, חלוקה של הזרם למרווחי QRS ויצירת אשכולות מקומית רצים ב job אחד של ה-MapReduce
- שני השלבים מתבצעים על זרם חדש של נתונים, ושלב ראשון יחסית פשוט ומהיר, לכן ניתן להריץ את שני השלבים ב job אחד.
- המיזוג מתפצל לשני job-ים, מיזוג והערכת אשכולות. שלב המיזוג רץ במקביל. במיזוג יש צורך לחשב את הקשר בין כל שני אשכולות, ולבצע מיזוג רק בחלק מהאשכולות. כדי להגדיל את המקביליות המיזוג מתפצל לשני שלבים, שלב התזמון ושלב המיזוג. שלב התזמון מחשב את הקשר בין כל שני אשכולות, ומתזמן את תוכנית המיזוג. המיזוג מתבצע רק עבור הנתונים להם קיימת תוכנית מיזוג. שלב התזמון מתבצע באופן סדרתי, שלב המיזוג רץ במקביל. לאחר המיזוג, איכות של כל אשכול מוערך מחדש, על מנת לבדוק האם יש צורך בפיצול האשכול. לא ניתן לשלב את הערכת אשכולות עם ה job הקודם, כי יש צורך לרוץ פעמיים על כל הנתונים, עבור מציאת מרכז של כל האשכול ועבור חישוב ממוצע רדיוס בהשוואה למרכז.
- פיצול גלובלי של האשכולות יתבצע ב job יחיד של MapReduce.
- לאחר שלב הפיצול האשכולות מוערכים מחדש ב job נפרד.

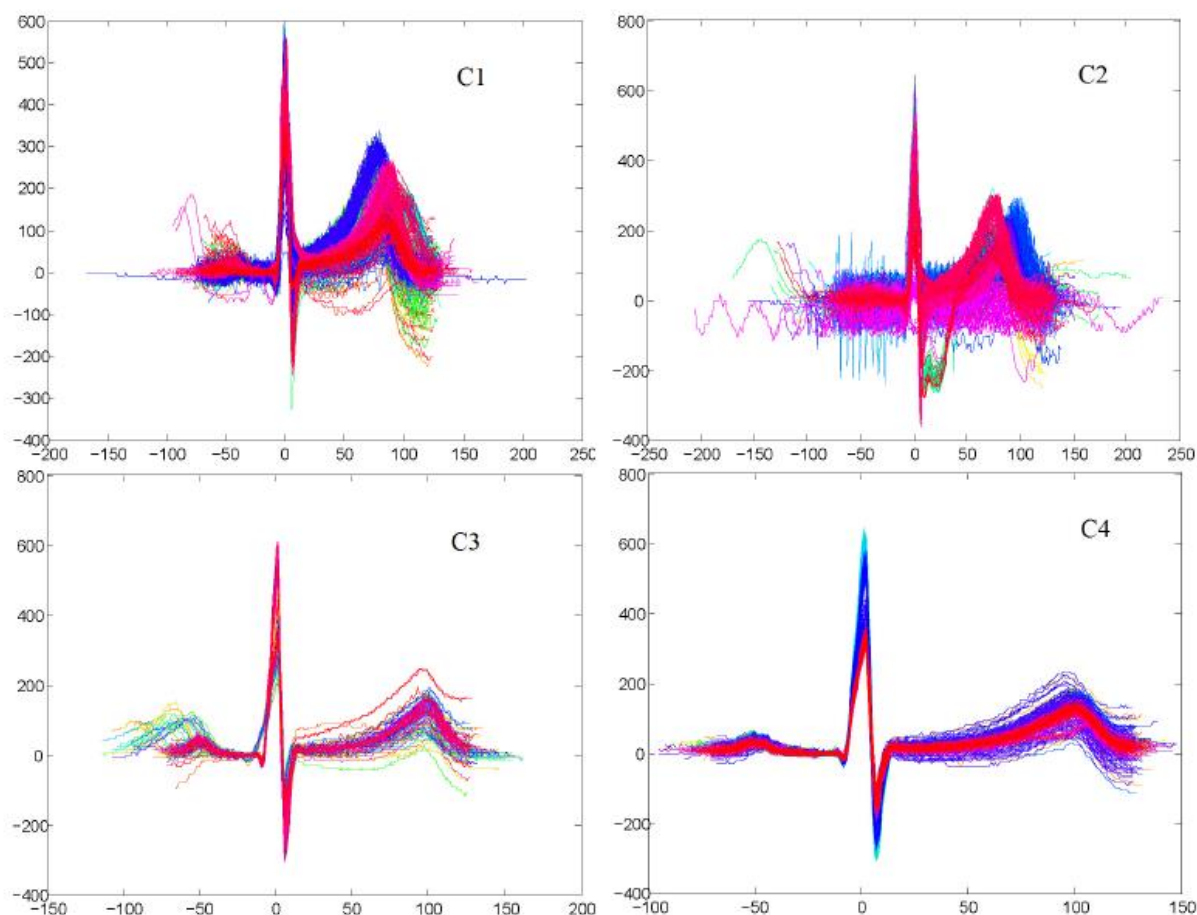
מקרה בוחן

המערכת נבדקה ב Hadoop 2.0, עם אשכול הומוגני בו כל צומת מכילה מעבד עם 4 ליבות, 4 GB RAM, 160 GB דיסק קשיח. הנתונים נאספו ממאות חולים, בעלי גילאים שונים, מין ומצב בריאותי. בבסיס נתונים קיימים יותר מ 1500 קבצי אקג, כל אחד מהם מכיל 2800 מרווחי QRS, סהכ 4,356,800 מרווחי QRS. הריצו את האלגוריתם על בסיס נתונים קיים, מהנתונים נוצרו ארבעה אשכולות. נתונים סטטיסטיים של כל אשכול ניתן לראות בטבלה 4.

ID	Cluster size	Total distance	Average radius	Maximal radius
C1	2753836	130889825.08	47.53	186.50
C2	1922412	107558951.40	55.95	202.10
C3	2003	55743.49	27.83	88.04
C4	11771	289684.31	24.61	110.13

טבלה 4 : נתונים סטטיסטיים של האשכולות

גודל האשכול הוא מספר מרווחי QRS באשכול, המרחק הכולל הוא סכום המרחקים בין מרכז האשכול למרווחי QRS באשכול. רדיוס ממוצע ורדיוס מקסימלי, הם ממוצע ומקסימום של מרחק בין מרכז האשכול למרווחי QRS באשכול. באיור 32 ניתן לראות תצוגה אינטואיטיבית של האשכולות המוצגים בטבלה 4.

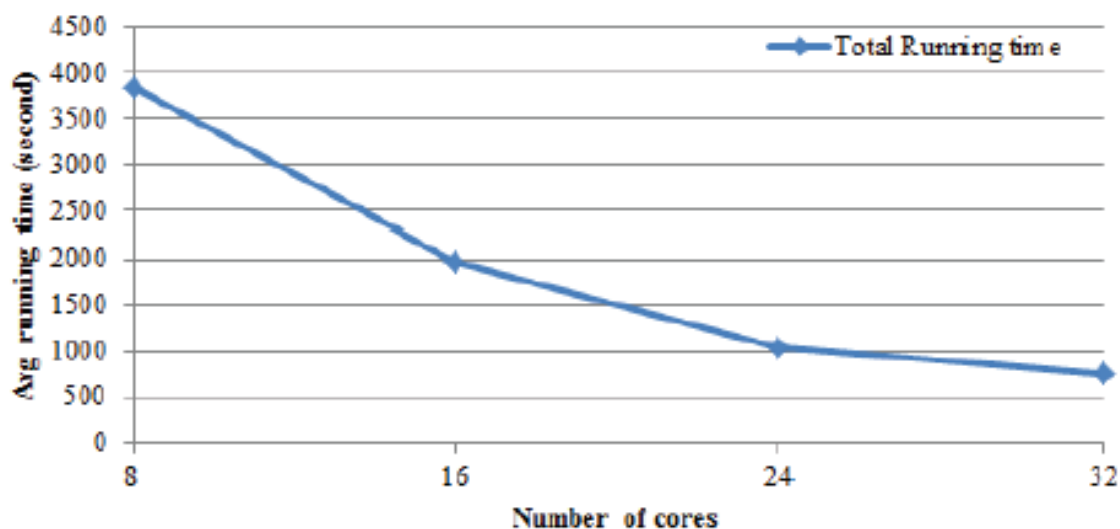


איור 32 – תצוגה אינטואיטיבית של האשכולות [28]

מסקנות: האיכויות של C1, C3, C4 הן טובות, המאפיין המורפולוגי של כל מרווח QRS בכל אשכול מותאם היטב. ב C2 קיימים רעשים, בדיקה מעמיקה הראתה שהדבר נוצר עקב שימוש לא נכון בחיישני לב.

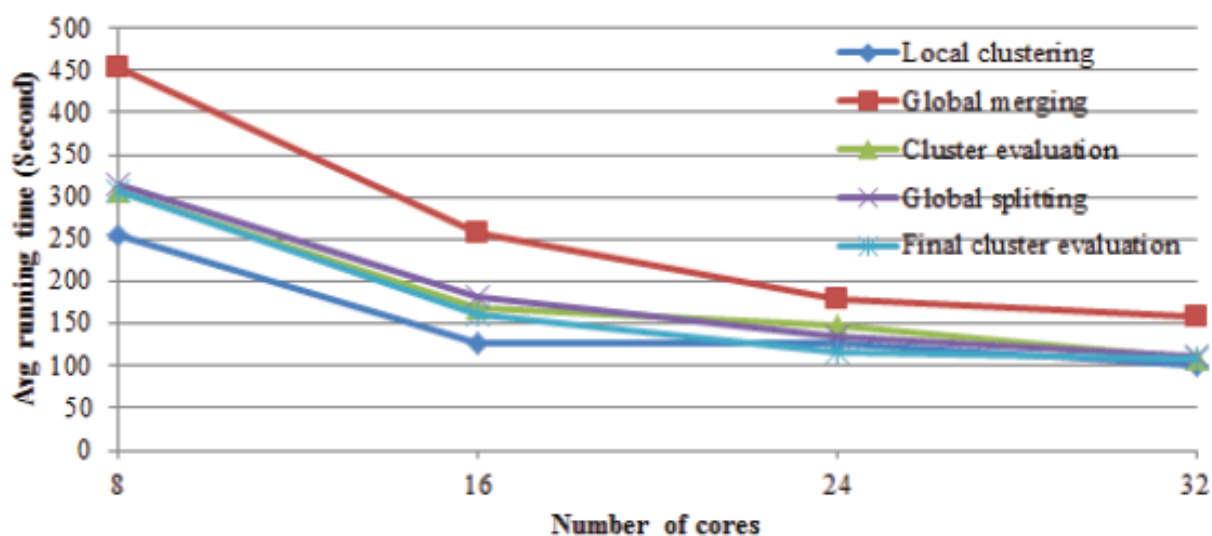
ניתוח האלגוריתם

באיור 33 מודגמים נתוני זמן ריצה ממוצע כתלות במספר הליבות. נמצא שזמן ריצה פוחת אקספוננציאלי כתלות במספר הליבות שנוספו



איור 33 – זמן ריצה ממוצע עבור מספר ליבות שונות [28].

באיור 34 ניתן לראות זמן ריצה של שלבי האלגוריתם.

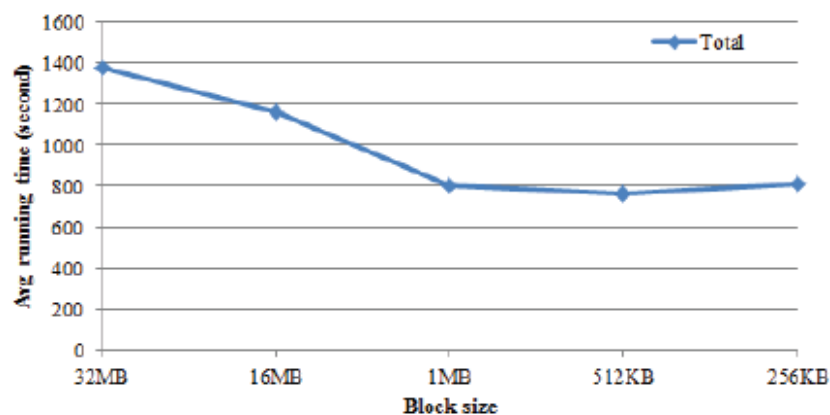


איור 34 - זמן ריצה ממוצע של שלבי אלגוריתם עבור מספר ליבות שונה [28]

שלב מיזוג לוקח את רוב הזמן במהלך הריצה בגלל ריבוי פעולות קלט/פלט במיזוג אשכולות.

האלגוריתם רץ על גודל שונה של בלוקים ב HDFS. באיור 35 ניתן לראות את הממצאים, הקטנת גודל הבלוק ב HDFS מ 32 מגה למגה משפר ביצועים. לא ניתן להקטין את גודל הבלוק מעבר למגה, בגלל שב HDFS כל

קובץ תופס בלוק (גם אם הוא קטן יותר מגודל הבלוק). אם מקטינים את הבלוק ב HDFS, יתקבלו קלט/פלט מיותרים ועומסים ברשתות, מצד שני, כל משימת map בדרך כלל רצה על בלוק, אם הבלוק קטן משמעותית מגודל הקובץ, יהיו יותר מדי משימות map משיגרום להאטה בביצועים.



איור 35 – תאור ביצועי מערכת כפונקציה של גודל הבלוק ב HDFS [28]

מסקנות

המערכת מבצעת כריית מידע על הנתונים בזמן אמת, על ידי הפעלת אלגוריתם הבונה אשכולות בשיטת מיזוג ופיצול אשכולות חדשים וקיימים. המערכת בונה אשכולות בצורה טובה, עם ביצועים מעולים על מספר רב של צמתים.

5 הצעה לשיפור מערכת TELE-NST

5.1 תאור מערכת TELE-NST

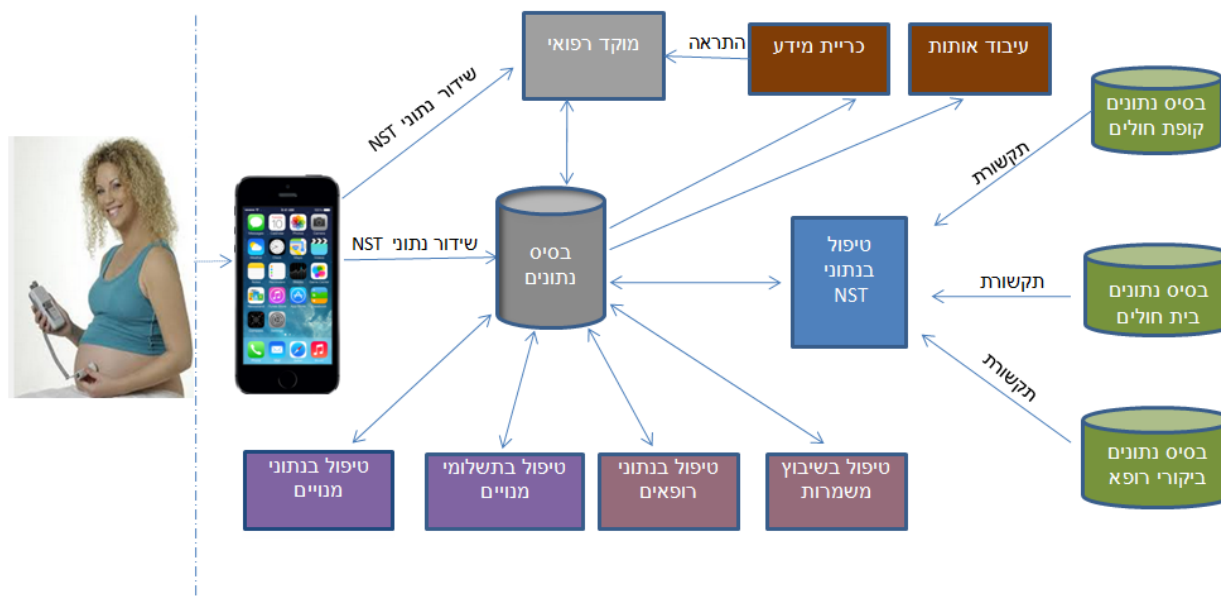
נתוני ניטור צירים וקצב לב עוברי יסומנו כ- NST. NST מתבצע במסגרת מרפאות ובתי חולים במהלך השבוע 25 עד שבוע 41 להריון. מטרת הניטור לזהות מוקדם ככל האפשר נשים המצויות בסיכון מוגבר ללידה מוקדמת ו/או מצוקה עוברית.

נשים שאובחנו כמצויות בסיכון גבוה ללידה מוקדמת מבצעות את הבדיקה בתדירות גבוהה ולעיתים הרופא מחליט לאשפז את האישה ההרה לניטור בתדירות גבוהה.

לאור חשיבות ביצוע בדיקת NST זמינה, הוחלט לפתח מערכת TELE-NST אשר תאפשר לבצע ניטור וניהול NST מרחוק באמצעות סלולר של קצב לב עוברי וצירים וזיהוי אוטומטי של נשים המצויות בסיכון מוגבר ללידה מוקדמת ומצוקה עוברית.

במסגרת הפרוייקט במדעי המחשב פותחה תת המערכת המאחסנת נתוני NST בענן, בבסיס נתונים רלציוני אורקל [30]. המערכת מקבלת את כל השידור של מנויה (המכיל נתוני NST), שומרת את השידור בבסיס נתונים ומפעילה תהליך של כריית מדיע על השידור. פותח ממשק וובי המאפשר לנהל את הנתונים של המנויות.

5.2 תאור ארכיטקטורת מערכת TELE-NST שפותחה בפרויקט מתקדם במדעי המחשב



איור 36 - רכיבים עיקריים של מערכת TELE-NST [30]

מערכת TELE-NST היא מערכת שרת לקוח. השרת מצוי בענן. הרכיבים העיקריים של המערכת מתוארים באיור 36 וכוללים:

לקוח (מובייל):

- אפליקציית אנדרואיד למובייל חושפת API מתאימים לשידור בו זמני של שני סוגי אותות-התכווצויות רחמיות וקצב לב עוברי (מכאן ואלך יכוננו נתוני NST) מהתקן ניטור ביתי והעברתם לבסיס נתונים מרכזי באמצעות הרשת הסלולרית.
- **מוקד רפואי** – נתונים הנקלטים מהמובייל משודרים בזמן אמת לשרת במוקד הרפואי. בעזרת ממשק וובי ניתן לבצע מספר פעולות שיפורטו בהמשך.

שרת בענן:

- **בסיס נתונים רלציוני** – המכיל נתוני NST, פרטי מנויות, פרטי רופאים ועוד. בסיס נתוני NST ישמר בענן
- **ממשק משתמש** – בעזרתו ניתן לבצע את הפעולות הבאות:
 - טיפול בנתוני מנויות – קליטת מנויה חדשה (בסוף התהליך יוגדר לכל מנויה מספר מנוי), עידכון פרטי המנויה, סגירת מנויה
 - טיפול בתשלומי מנויות
 - טיפול בנתוני רופאים – קליט, עדכון, סגירת פרטי רופא
 - טיפול בשיבוץ משמרות רופאים אוטומטי
 - צפייה מקוונת בנתוני שידור NST נוכחי ובשידורים קודמים של מנויה
 - צפייה ממוקדת במקטעי שידור NST חשודים (ZOOM)
 - השוואה איכותית וכמותית בין שידורי NST שונים למנויה
 - כריית מידע בזמן אמת על נתוני השידור הנקלט והתראה בזמן אמת במקרה ונמצאו ממצאים חריגים
 - עיבוד אותות NST

בסיסי נתונים חיצוניים – התממשקות לשלושה בסיסי נתונים חיצוניים קופת חולים, בית חולים, ביקור רופא וקבלת תוצאות עדכניות של בדיקות הכלליות שבוצעו במהלך מעקב אחר ההריון.

5.3 נתוני מערכת TELE-NST

נתוני המערכת מורכבים מנתונים בדידים ונתונים רציפים. נתונים בדידים מכילים מידע כללי אודות הרופאים והמנויות, נתוני בדיקות שהמנויה ביצעה (נתונים הנשלפים מבסיסי נתונים חיצוניים), ותוצאות ניתוח נתוני NST.

נתונים רציפים, הם סדרה רציפה בזמן, בכל שידור יש 2 סדרות אחד של קצב לב עוברי (FHR) ואחד של צירים (UC). הערכים שלהם הולכים בזוגות Y של FHR אחרי Y של UC באותה נקודת זמן (כל זוג ערכים מגיעה כל רבע שניה).

נתוני מערכת נשמרים עם סכמה מובנית בענן, בבסיס נתונים אורקל. הנתונים הרציפים נשמרים לכל שידור של מנויה, בסוג נתונים CLOB (character large object).

5.4 תאור הבעיות במערכת TELE-NST

המערכת הנוכחית נשמרת בענן, בבסיס נתונים רלציוני ומבצעת עיבוד על נתוני NST לאחר קבלת כל השידור. דרישות המערכת, לגדול ללא הגבלה בבסיס נתונים, ולנתח נתונים בזמן אמת במהירות וביעילות. במערכת הנוכחית קיימות 2 בעיות עיקריות:

1. בעיית ממדים – גידול בהיקף ובנפח הנתונים שישמרו וכן שישודרו, יגרום לקריסת בסיס נתונים (רוב בסיסי נתונים רלציונים לא תומכים בהרחבה לצמתים נוספים).
2. בעיית עיבודים - כריית מידע בזמן אמת, מצריכה שליפה מהירה ויעילה של נתונים. בסיס נתונים רלציוני עם נפח גדול, לא ידע להתמודד עם המשימה. בנוסף, RDBMS לא יכול להתמודד עם כמות גדולה של OLTP (עיבוד טרנזקציות מכוון) באותו הזמן.

5.5 הצעה לשיפור מערכת TELE-NST

- בסיס הנתונים הקיים במערכת TELE-NST חייב לתמוך בקריטריונים הבאים:
- במערכת קיימים נתונים רפואיים כגון פרטי מנויות, פרטי, תשלום וכ"ד, בהם חשוב דיוק במידע, היות והנתונים מוצגים במערכת הוויבית, לכן בסיס נתונים חייב לתמוך ב ACID.
 - יכולת הרחבה של בסיס נתונים ללא הגבלה
 - כריית מידע ירוץ בזמן אמת, לכן השמירה ואחזור של הנתונים צריכים להיות מאוד מהירים ויעילים

מכאן, בסיס נתונים מצד אחד צריך לתמוך ב ACID ומצד שני, אחזור ושליפה חייבים להיות מהירים. בנוסף בסיס נתונים יכול לגדול לממדים עצומים. הבסיס נתונים המתאים הוא NewSQL שיתמוך גם ב ACID וגם בביצועים של NoSQL הכוללים יכולת הרחבה ללא הגבלה (מוסבר בסעיף 3.3). NewSQL לא צלח, עדיין מחפשים לו פיתרון. מכאן שיש לפצל את בסיס הנתונים בהתאם למטרות השונות. עבור התמיכה ב ACID נשתמש בבסיס נתונים רלציוני Oracle, ועבור תהליך של כריית מידע בזמן אמת נוסיף מחסן מידע שיתמוך ב NoSQL ויכולת הרחבה ללא הגבלה.

הצעה לשיפור

בסיסי נתונים וממשקים ב Advanced TELE-NST

- בפרויקט קיים בסיס נתונים רלציוני Oracle בענן, המכיל את כל נתוני המערכת. בסיס הנתונים ישאר ללא שינוי סכמה, כל הנתונים הקיימים בבסיס נתונים חייבים להיות מדויקים, כולל השידורים של המנויות (במערכת הוובית, שולפים את השידור של המנויה יחד עם הפרטים שלה). בסיס נתונים יכול לגדול, לכן בבסיס נתונים יישמרו רק הנתונים של חצי שנה האחרונה (נשים משדרות נתוני NST החל משבוע 25, לכן הנתונים של המנויות הפעילות יהיו זמינים בבסיס נתונים). שאר הנתונים יישמרו בתור גיבוי במכונות אחרות (כל חצי שנה ירוץ תהליך שייפלט ויגבה את בסיס הנתונים). במקרה והפילטור של חצי שנה אחורה לא יהיה מספיק טוב, ניתן להשתמש ב Oracle Rac (מוסבר בסעיף 3.3.2) התומך בהרחבה של בסיס נתונים לצמתים נוספים, הבעיה שהפיתרון הזה מאוד יקר וקיימת מורכבות רבה בהתקנה ותחזוקה (עבור שמירה על קשרים בין הטבלאות, הנתונים מפוזרים בין כל הצמתים)
- בניית מחסן נתונים מעל Hadoop (גרסה חדשה של Hadoop הכוללת את Yarn לשיפור הביצועים) כך שהנתונים יישמרו במערכת קבצים מבוזרת HDFS. מעל שיכבת ה- Hadoop נתקין את Hbase, בסיס נתונים NoSQL המתאים לעבודה ב- Real Time, על ידי כך שמאפשר גישה רנדומלית לקריאה/כתיבה (מוסבר בסעיף 3.4.3). במחסן נתונים יישמרו כל הנתונים הרלוונטיים לכריית מידע, כולל היסטוריית השידורים של המנויות.
- לביצוע אנליטי של הנתונים יש להתקין את Hive, ממשק מעל HBase עם שפה דמויית SQL אשר מתורגם בזמן הריצה לתוכניות שונות של MapReduce.

תהליך כריית מידע:

המערכת תבצע כריית מידע בזמן אמת, על ידי מיזוג ופיצול אשכולות בצורה דינמית בדומה למערכת המתוארת בסעיף 4.4. המערכת תבצע כריית מידע מול מחסן נתונים, במטרה להשתמש ב Hadoop MapReduce, ולחלק את החישוב לכמה שיותר צמתים להשגת ביצועים טובים. אלגוריתם של כריית מידע יהיה בנוי מארבע שלבים, בכל שלב הנתונים ישמרו מקומית במחשב, לאחר סיום כל השלבים הנתונים ישמרו ב HDFS.

שלבי האלגוריתם (ניתן לראות את ההסברים בסעיף 4.4):

- חלוקת הזרם למרווחים
- יצירת אשכולות מקומית (אלגוריתם k means)
- מיזוג אשכולות גלובלי (אשכולות בהן קיים קשר ימוזג, בין כל האיטרציות של הנתונים)
- פיצול אשכולות גלובלי (אשכולות עם עיוותים יפוצלו לקטנים יותר)

כל ארבעת השלבים של האלגוריתם, יפוצלו ל 5 שלבים ב MapReduce. איור 31 מציג את החלוקה.

שלבי ה MapReduce (כל שלב הוא job נפרד, הסברים ניתן למצוא בסעיף 4.4):

- חלוקה למרווחים ויצירת אשכולות
- מיזוג אשכולות
- הערכת אשכולות מחדש (איכות אשכול נמדדת לפי שגיאת עיוות, מעריכים אשכולות כדי לבדוק האם יש צורך בפיצול אשכול).
- פיצול גלובלי (פיצול אשכולות בעזרת אלגוריתם k means)
- הערכת אשכולות

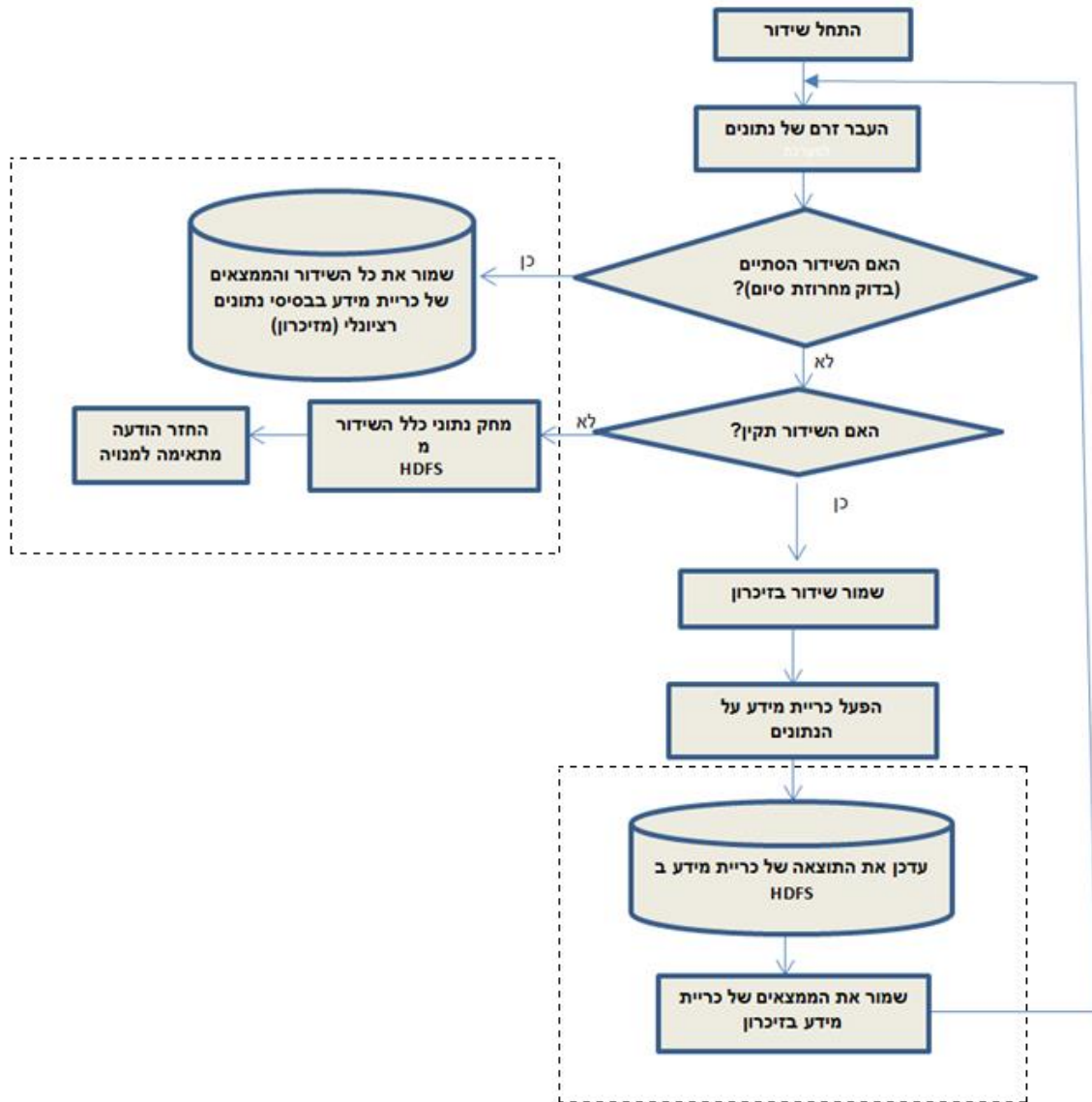
צד טכני בפרויקט

לבסיס נתונים קיים Oracle מתווסף מחסן נתונים, נתאר את הנתונים והתהליך עידכון נתונים במחסן נתונים.

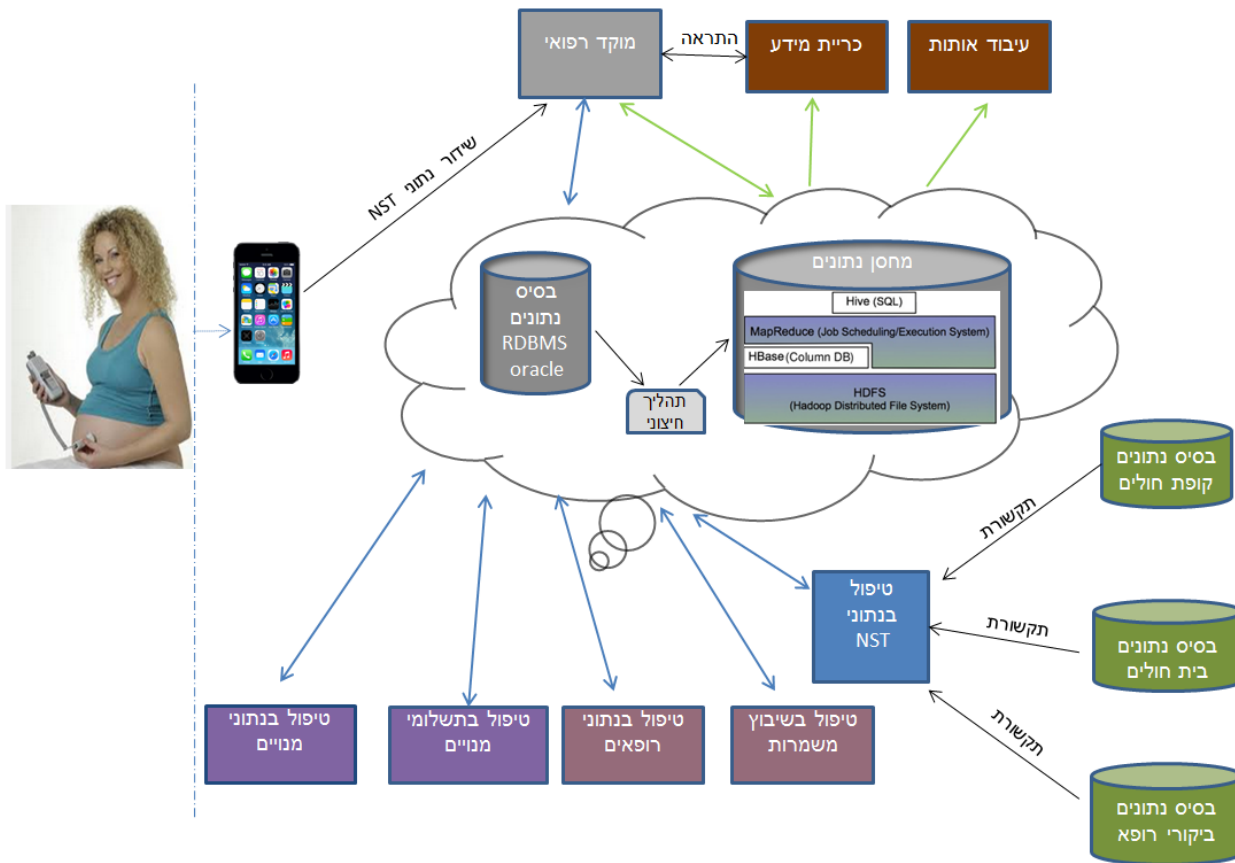
- המנויה רוכשת את המכשיר הביתי ונרשמת במערכת, הנתונים שלה יישמרו בבסיס נתונים רלציוני (oracle) והנתונים הרלוונטים במחסן נתונים (נתונים רלוונטים לכריית מידע, כגון גיל, תאריך ווסת האחרון וכ"ד), כך שבעת כריית מידע לא יהיה צורך לגשת לבסיס נתונים הרלציוני.
- כדי לא להכביד על המערכת הנוכחית, בזמן שידור, במחסן מידע יישמרו רק הנתונים של כריית מידע (כל הנתונים הרלוונטיים לאלגוריתם של כריית מידע), כל שאר הנתונים יישמרו בבסיס נתונים רלציוני (השידור המלא של המנויה, ההמלצה, עידכונים של ההמלצה וכ"ד). תהליך חיצוני (ראה בהמשך) יעביר את שידור המלא של המנויה מבסיס נתונים רלציוני למחסן נתונים.
- מחסן נתונים, יכיל את היסטוריית השידורים של מנויה, וכל הפרטים הרלוונטיים לכריית מידע.

- שידור המנויות, תופס הרבה מקום, לכן השידור יישמר בבסיס נתונים רלציוני במשך X זמן (X פרמטר זמן בשעות)
- הקשר בין מחסן נתונים לבסיס נתונים רלציוני, יהיה בעזרת מזהה יחודי של השידור (transmission_id). השדה קיים בבסיס נתונים רלציוני, בטבלה NST (בעזרת המזהה ייחודי ניתן להגיע לשידור של הלקוחה, ולפרטי מנוי של הלקוחה).
- לבסיס נתונים רלציוני, תתווסף טבלה חדשה, שתכיל את השינויים של המערכת, הרלוונטים למחסן נתונים, לדוגמא, אם הרופא מחליט לשנות המלצה, הטבלה תתעדכן. הטבלה תכיל את השדות הבאים :
Pk - מפתח ראשי (מספר רץ)
Rec_type – סוג רשומה, לדוגמא המלצה.
Description – יכיל את השינויים (לדוגמא בהמלצה את מפתח ההמלצה והתאור).
Created_date – תאריך יצרית הרשומה
- יתווסף תהליך חדש, שיבדוק באופן קבוע אם היו שינויים בטבלה החדשה (כל Y זמן), אם כן, ייעדכן את מחסן הנתונים בהתאם. התהליך החדש ירוץ בשרת אחר משרת בו רצה המערכת, כדי לא להכביד על המערכת.

באזור 37 מוצג תרשים זרימה לשידור נתוני NST. שידור נתוני NST מתחיל מיד לאחר זיהוי המנויה במערכת. כל עוד לא הסתיים השידור, השידור נשמר בזיכרון (שרת בענן), ומופעל בזמן אמת תהליך כריית מידע על השידור. בזמן ביצוע האלגוריתם של כריית מידע, הנתונים נשמרים ב HDFS. בסיום השידור, השידור המלא והממצאים של כריית מידע נשמרים בבסיס נתונים רלציוני, ובהמשך באמצעות תהליך חיצוני, מועברים למחסן נתונים.



איור 37 – שידור נתוני NST במערכת Advanced TELE-NST



איור 38 – ארכטיקטורה מערכת Advanced TELE-NST

באיור 38, החיצים הירוקים מתחברים למחסן מידע, החיצים הכחולים לבסיס נתונים רלציוני. תהליך של כריית מידע ועיבוד אותות רצים מעל מחסן נתונים. המנויה משדרת צירים דרך התקן סוללרי, על הצירים בזמן אמת מופעל תהליך של כריית מידע. בזמן ביצוע כריית מידע, הנתונים נשלפים ומעודכנים במחסן נתונים. תוצאות כריית מידע והשידור המלא נשמרים בבסיס נתונים רלציוני. קיים תהליך חיצוני, הבדק את בסיס נתונים הרלציוני באם היו שינויים, ומעדכן את מחסן נתונים בהתאם. כל נתוני המערכת הוובית, נשלפים מבסיס נתונים רלציוני.

במסגרת פרויקט מתקדם של לימודים לתואר שני, פותחה מערכת TELE-NST המאפשרת לבצע ניטור וניהול נתוני צירים וקצב לב עוברי, מרחוק באמצעות סלולר וזיהוי אוטומטי של נשים המצויות בסיכון מוגבר ללידה מוקדמת ומצוקה עוברית. ניטור צירים וקצב לב עוברי (NST- Nonstress Test) מבוצע במסגרת מרפאות ובתי חולים במהלך השבוע 25 עד שבוע 41 להריון. מטרת הניטור לזהות מוקדם ככל האפשר נשים המצויות בסיכון מוגבר ללידה מוקדמת ו/או מצוקה עוברית. בדרך כלל משך זמן שידור הינו 20 דקות, תדירות השידור מספר פעמים ביממה לאשה נבדקת, קצב השידור הוא כל רבע שניה. המערכת מאפשרת לבצע ניטור NST מרחוק באמצעות סלולר, מעבירה את השידור המלא של המנויה לענן, שומרת את השידור בבסיס נתונים רלציוני, ומפעילה תהליך של כריית מידע על השידור. אחת הבעיות של המערכת היתה להתמודד עם מידע רב, ככל שבסיס נתונים גדל, היתה האטה בזמן עיבוד נתונים עד שהמערכת בסופו של דבר קרסה. בעיה נוספת, היא שכריית מידע לא מתבצעת בזמן אמת (העיבוד מתבצע על נתונים לאחר קבלת כל השידור, קיים צורך במענה מיידי לאירועי צירים וקצב לב עוברי). לפיכך החלטתי להמשיך ולחקור במסגרת עבודה מסכמת גישות שונות לאיחסון נתונים המועברים ממובייל לענן.

בתחילת העבודה נסקרו הטכנולוגיות מובייל (מעה"פ אנדרואיד), מחשבו ענן, שילוב בין מובייל לענן בתחום הרפואי. ראינו שמחשוב מובייל בענן בתחום הרפואי, מעודד את השימוש בשירותי רפואה מרחוק ומפחית את הבעיות השכיחות ביישומים רפואיים כגון איחסון פיזי קטן, מהירות חישוב, אבטחה ופרטיות. לאחר מכן, הוצג נושא ה BigData, איך מתמודדים איתו במהירות וביעילות ופיתרונות הרבים ל Big Data בענן. העבודה מתמקדת בבסיס נתונים NoSQL, במנועים העיקריים לטיפול ב – NoSql, ובהבדל המהותי בין בסיס נתונים רלציוני ל NoSql. ראינו שבענן ניתן להשתמש בבסיס נתונים רלציוני ו NoSQL לשמירת מידע. כאשר מדובר במערכות בהן הדיוק קריטי, כגון נתונים רפואיים, חשבונאות, לוגיסטיקה כדאי להשתמש בבסיס נתונים רלציוני התומך ב ACID. מצד שני בסיס נתונים מבוזר כגון NoSQL, התומך בביצועים טובים עם ארכיטקטורה מבוזרת הניתנת להרחבה, הוא פיתרון פשוט להבנה ולתחזוקה, ופיתרון טוב במקרים של ניתוחים סטטיסטיים, או מנועי חיפוש באינטרנט בהן הפיתרון תקין גם אם לא כל השורות מוחזרות. בשלב האחרון של העבודה המסכמת, נסקרה פלטפורמת ה- Hadoop. פלטפורמת Hadoop היא אחד הפתרונות הבולטים בארגונים לטיפול ב- Big Data בענן, המספקת תשתית לפרויקטים עם בסיס נתונים מסוג NoSQL (HDFS), ותשתית מבוזרת לעיבוד מקבילי (Hadoop MapReduce).

אחד מהתוצרים של העבודה המסכמת הנוכחית היא המלצה להמשך פיתוח פרויקט TELE-NST, שבו יבנה מחסן מידע עם בסיס נתונים NoSQL מעל Hadoop לאחסון נתוני NST (בסיס נתונים ניתן להרחבה, עם ביצועים מהירים), וביצוע כריית מידע בזמן אמת במהירות ויעילות באמצעות האלגוריתם למיזוג ופיצול אשכולות בצורה דינמית. יש לציין שהחישובים לבניית האשכולות, יבוצעו במקביל על מספר רב של צמתים באמצעות Hadoop MapReduce.

כמו כן, ניתן להמשיך בחקר ה-Big Data, בכיוונים הבאים :

- פיתוח כלים לסטנדרטליזציה של מגוון פתרונות ל Big Data כולל פיתוח API למעבר קל ויעיל בין פתרונות שונים.
- פיתוח כלי המספק פיתרון ל Big Data ע"פ דרישות הארגון.
- פיתוח כלים למעבר מבסיס נתונים רלציוני ל – NoSQL.
- פיתוח פיתרון יעיל יותר ל NewSQL, כך שמצד אחד יתמוך ב ACID ומצד שני יגיע לאותם הביצועים של NoSQL (כידוע פיתרון ה NewSQL לא צלח).

- [1] Ahmed E. Youssef (2014) "**A framework for secure healthcare systems based on big data analytics in mobile cloud computing environments**" International Journal of Ambient Systems and Applications (IJASA) Vol.2, No.2
- [2] Alzahrani A, Alalwan N., SarrabM "**Mobile Cloud Computing: Advantage, Disadvantage and Open Challenge**" EATIS '14 Proceedings of the 7th EuroAmerican Conference on Telematics and Information Systems Article No. 21 ACM New York, NY, USA 2014
- [3] Chandar J., (2010) "**Join Algorithms using Map/Reduce**" Master's Thesis, Informatics MSc, School of Informatics, University of Edinburgh
- [4] Christian Vecchiola, Suraj Pandey, Rajkumar Buyya "**High-Performance Cloud Computing: A View of Scientific Applications**" Cloud computing and Distributed Systems. Laboratory Department of Computer Science and Software Engineering The University of Melbourne, Australia, ISPAIEEE Computer Society 2009 , p. 4-16
- [5] Condie T., Conway N., Alvaro P., Hellerstein J. (2010) "**MapReduce Online**" Proceedings of the 7th USENIX conference on Networked systems design and implementation Pages 21-21
- [6] Hoang T. Dinh, Chonho L., DusitN., and Ping W. (2011) "**A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches**" Wireless communications and Mobile Computing Issue 18 Pages i–ii, 1587–1692
- [7] Jianye L., Jiankun Y. (2011), "**Research on Development of Android Applications**" Intelligent Networks and Intelligent Systems (ICINIS)
- [8] Khullar Y., Raj S., Neeraj S. (2013) "**Impact of Cloud Computing on Healthcare**" International Journal of Modern Engineering & Management Research
- [9] Kutzer M., Feizipour C. "**Communication and Modeling of Simple Robots**" Wireless Security ,Computer Articles Online. 2007
- [10] Lan M., Samy L., Alshurafa N., Sua M., Ghasemzadeh H., Sarrafzadeh M. (2012) "**WANDA: An End-to-End Remote Health Monitoring and Analytics System for Heart Failure Patients**" Proceedings of the conference on Wireless Health
- [11] Liu M. 2013, "**A Study of Mobile Sensing Using Smartphones**" Hindawi Publishing Corporation International Journal of Distributed Sensor Networks
- [12] Lutz Schubert, (2010) "**The Future Of Cloud Computing**" Report from EC CLOUD computing Expert Group, Pew Research Center's Internet & American Life Project & Elon University, pages:1 -26
- [13] Murthy A., Markham J., Vavilapalli V., Eadline D. (2014) "**Apache Hadoop Yarn**" SOCC 13 Proceedings of the 4th annual Symposium on Cloud Computing, Article No. 5
- [14] Niroshinie F., Seng W. Loke, WennyR. (2012) "**Mobile cloud computing: A survey**" Department of Computer Science and Computer Engineering, La Trobe University, Australia.
- [15] Pocatilu P., Boja C., Ciurea C. (2013) "**Syncing Mobile Applications with Cloud Storage Services**" Department of Economic Informatics and Cybernetics The Bucharest University of Economic Studies
- [16] Pooja S. Honnutagi, (2014) "**The Hadoop distributed file system**" Computer Science and Information Technologies, Vol. 5 (5) , 2014, 6238-6243
- [17] Pramod Kulkarni A., Khandewal M. (2014) "**Survey on Hadoop and Introduction to YARN**" International Journal of Emerging Technology and Advanced Engineering Volume 4, Issue 5.
- [18] Rodrigo N., Assun M., Calheiros R., Bianchi S, (2013) "**Big Data Computing and Clouds: Trends and**

Future Directions” *Parallel and Distributed Computing* , Volumes 79-80, pages 3-15, 2015

- [19] Sakr S., Liu A., Ayman G. F (2013) **“The Family of MapReduce and Large-Scale Data Processing Systems”** *ACM Computing Surveys*, Volume 46 Issue 1 Article No. 11
- [20] Salunkhe D., Bahirat B., Koushik N.V, Javale D (2014) **“Study of Hadoop Features for Large Scale Data”** *International Journal of Innovative Technology and Exploring Engineering*
- [21] Santhosh K. Gajendran **“A Survey on NoSQL Databases”** Survey, University of Illinois, College of Engineering, 1998.
- [22] Shortliffe H.E, Octo Barnett G. (2006) **“Biomedical Informatics, computer applications in health care and biomedicine”** *Medical informatics: computer applications in health care* Pages 37-69
- [23] Somasundaram M., Gitanjali S., Govardhani T.C, Lakshmi Priya, RG.. Sivakumar (2011) **“Medical Image Data Management System in Mobile Cloud Computing Environment”** *International Conference on Signal, Image Processing and Applications With workshop of ICEEA*
- [24] Srinivas T, Patil A, Aswhwini A, Ranjana R (2015) **“A Mobile Cloud based Approach for Secure Medical Data Management”** *International Journal of Computer Applications (0975 – 8887) Volume 119 – No.5*
- [25] Vijiyalakshmi V., Akila A., Nagadivya S. (2012) **“The Survey On MapReduce”** *International Journal of Engineering Science and Technology*
- [26] Wandelt S., Rheinländer A., Bux M., Thalheim L., Haldemann B., Leser U. (2012) **“Data Management Challenges in Next Generation Sequencing”** *Datenbank-Spektrum Volume 12, Issue 3 , pp 161-171*
- [27] Wynden R., Sun Y, Nguyen A. V, (2011) **“HBase, MapReduce, and Integrated Data Visualization for Processing Clinical Signal Data”** *Computational Physiology from the AAAI Spring Symposium*
- [28] Yang I, Zhang J., Zhang Q., (2013) **“A parallel system for clustering ECG streams based on MapReduce”** *Global Communications Conference (GLOBECOM), IEEE*
- [29] Yoon-Joon L, Kyong-Ha L., Hyunsik C., YonDohn C., Bongki M. (2011) **“Parallel Data Processing with MapReduce: A Survey”** *ACM SIGMOD Record archive Volume 40 Issue 4 Pages 11-20*
- [30] Kaykov L., Herman M. (2015) **“TELE-NST system planning, focusing on NST data”** *Open University*
- [31] Apache Hadoop Yarn – Background & Overview <http://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/>
- [32] Difference between Hadoop and RDBMS <http://www.wikidifference.com/difference-between-hadoop-and-rdbms/>
- [33] Hadoop When to Use What <http://smartdatacollective.com/mtariq/120791/hadoop-toolbox-when-use-what>
- [34] Hadoop Yarn <http://www.slideshare.net/arnonrgo/hadoop-yarn-35157325?ref=http://arnon.me/2014/05/hadoop-yarn-overview>
- [35] Cloud Computing Trends [http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2015-state-cloud-survey#Hybrid Cloud Remains the Preferred Strategy](http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2015-state-cloud-survey#Hybrid%20Cloud%20Remains%20the%20Preferred%20Strategy)
- [36] NoSQL, NewSQL, or RDBMS: How To Choose <http://www.informationweek.com/big-data/big-data-analytics/nosql-newsql-or-rdbms-how-to-choose/a/d-id/1297861>
- [37] Comparing the Hadoop Distributed File System (HDFS) with the Cassandra File System (CFS) <http://www.datastax.com/wp-content/uploads/2012/09/WP-DataStax-HDFSvsCFS.pdf>
- [38] Introduction to Hadoop http://www.cs.unm.edu/~estrada/files/08-intro_to_hadoop.pdf
- [39] MongoDB, Cassandra, and HBase -- the three NoSQL databases to watch <http://www.infoworld.com/article/2848722/nosql/mongodb-cassandra-hbase-three-nosql-databases-to-watch.html>
- [40] Choosing the right NoSQL DB <http://www.slideshare.net/EdurekaIN/no-sql-databases-35591065>

- [41] System properties comparison <http://db-engines.com/en/system/Cassandra%3BHBBase%3BMongoDB>
- [42] Cloud risks <http://www.zdnet.com/article/saas-paas-and-iaas-three-cloud-models-three-very-different-risks/>
- [43] Google BigTable <http://searchdatamanagement.techtarget.com/definition/Google-BigTable>

Abstract

Fetal Non-Stress Test (NST) performs in primary and hospitals during week 25 to 41 of pregnancy. The purpose of the monitoring is to detect early as possible women at increased risk of premature birth and/or fetal distress. As part of Advanced Computer Science project, developed subsystem for handling NST data (part of TELE-NST system). The system via cellular allows NST monitoring and remote management, delivers full subscriber's transmission to the cloud, saves the transmission in relational database and triggers data mining process. One of the major system problems is difficulty to deal with growing data. As the database grows, there are slowdowns during data processing, as a result the system crashes. Another problem, data mining for NST data is not performed in real time (the processing is done after receiving all data transmission, while there is a need for an immediate response to the events of fetal non stress test).

The work overviews mobile technologies (android operating system), cloud computing and mobile cloud integration in medical field. In addition, the work overviews Big Data, ways to deal with it quickly and effectively and provides variety of Big Data solution over the cloud.

This work focuses on NoSQL databases and in major NoSQL engines. Additionally, the work addresses the fundamental differences between NoSQL and relational database. Finally, the work reviews Hadoop MapReduce storage method.

One of the major final work recommendations is to continue TELE-NST development and to build data warehouse in the cloud above Hadoop for storing NST data. In addition, perform real time data mining implementation using dynamically merging and splitting clusters algorithm, while calculations for building clusters will be performed in parallel on multiple nodes using Hadoop MapReduce.

Table of Contents

Abstract	5
1 Introduction	6
1.1 Mobile	7
1.1.1 Introduction.....	7
1.1.2 Android Architecture	7
1.1.3 Interface Types	9
1.1.4 Mobile Sensors.....	10
1.2 Cloud Computing	11
1.2.1 Introduction.....	11
1.2.2 Architecture.....	12
1.2.3 Service Models.....	13
1.2.4 Deployment Models.....	15
2 Mobile Cloud Computing.....	16
2.1 Introduction	16
2.2 Architecture.....	18
2.3 Syncing Mobile Applications with Cloud Storage Services	19
3 Medical Mobile Cloud Data Storage.....	21
3.1 Medical Data Types.....	21
3.2 Medical Data Issues	21
3.2.1 Data Volume	21
3.2.2 Privacy Security and Confidentiality	22
3.3 Databases	23
3.3.1 NoSQL	23
3.3.2 RDBMS, NoSQL and NewSQL comparison	24
3.3.3 NoSQL Major Engines.....	26
3.4 MapReduce.....	30

3.4.1	Introduction.....	30
3.4.2	Programming Model	30
3.4.3	MapReduce - Advantages and Disadvantages	36
3.4.4	Implementations.....	38
3.4.5	Versions and Improvements above MapReduce	40
3.5	Hadoop	42
3.5.1	Introduction.....	42
3.5.2	HDFS	43
3.5.3	Hadoop MapReduce.....	47
3.5.4	Job Scheduling	52
3.5.5	YARN	53
3.5.6	HBASE	58
3.5.7	RDBMS and Hadoop comparison.....	59
4	Literature Review	60
4.1	Introduction	60
4.2	Medical Image Management system in Mobile Cloud.....	60
4.3	Wanda, Health Monitoring System for Heart Failure Patients	62
4.4	Clustering ECG streams in the Cloud	67
5	TELE-NST System Improvement proposal.....	76
5.1	TELE-NST System description	76
5.2	TELE-NST Architecture.....	76
5.3	System Data	78
5.4	TELE-NST problems description	78
5.5	Improvement proposal	78
6	Summary, Conclusion and Future Directions.....	84
7	Reference	86

The Open University of Israel
Department of Mathematics and Computer Science

Method for storing NST data transferred from mobile to cloud

Thesis submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Computer Science Division

By
Larisa Kaykov
304684418

Prepared under the supervision of Dr. Maya Herman
March 2016