האוניברסיטה הפתוחה

המחלקה למתמטיקה ומדעי המחשב

עבודה מסכמת לתואר שני במדעי המחשב בנושא:

סקירת אבטחה במסדי נתונים לא רלציונים

עבודה מסכמת זו הוגשה כחלק מהדרישות לקבלת תואר

"מוסמך למדעים" M.Sc. במדעי המחשב

באוניברסיטה הפתוחה

המחלקה למתמטיקה ומדעי המחשב

נובמבר 2012

מגיש: ליאור אוקמן, ת.ז. 033480872
מנחה: פרופ' אהוד גודס

תוכן עניינים

רשימת איורים

רשימת טבלאות

1 תקציר

בשנת 2000, בכנס בנושא מערכות מבוזרות, הציג Eric Brewer תאוריה [9], לפיה בבניית מערכות מבוזרות אסינכרוניות ניתן לקיים רק שתיים משלוש התכונות Consistency, Availability ו־Persistency. מערכות מסדי נתונים רלציונים המשמשים מערכות מבוזרות המקימים את מודל ACID זוהו כרכיבים שאינם מבטיחים את תכונת ה־Availability ולכן חווית המשתמשים בסביבות מבוזרות נפגעות. בעקבות זאת, החלו להופיע מסדי נתונים לא רלציונים, שתוכננו כך שמפתחי המערכות יוכלו לבחור את שתי התכונות החשובות לאפליקציה שהם מפתחים. מסדי נתונים אלה תוכננו כך שהן מוותרות על חלקים מהיכולות של מסדי נתונים רלציונים כדי לאפשר ביזוריות גדולה יותר, או ביצועים טובים יותר. לדוגמה, במסדי נתונים אלה מודל הנתונים איננו נקבע מראש, אלא משתנה בהתאם לצורכי האפליקציה המשתמשת במסד הנתונים, או שאינם תומכים בשפת SQL לביצוע שאילתות. מכיוון שמסדי הנתונים האלה הופיעו בראשונה כבסיסי נתונים של אתרי אינטרנט, הם מכילים או משתמשים בשפות וטכנולוגיות מתחום בניית אתרים, כגון JSON לתיאור נתונים ו־JavaScript או שפות תכנות דומות כשפות שאילתות, ולכן כונו בשם NoSQL. מכיוון שמדובר במודל מבוזר, בדר"כ מסדי נתונים אלה מאפשרים שימוש באלגוריתם MapReduce כדי לשלוף נתונים בצורה יעילה ומבוזרת [10].

ניתן לאפיין אבטחה במערכות בסיסי נתונים רלציונים בארבע דרישות עיקריות במודל המכונה CIAA [11, 12]:

1. סודיות Confidentiality ־ **C**אסור שמשתמש לא מורשה יוכל לגשת לנתונים
2. שלמות Integrity ־ **I** רק משתמשים מורשים יכולים לעדכן מידע
3. זמינות Availability ־ **A**משתמש יכול לבצע כל פעולה שיש לו הרשאות לבצעה
4. אימות Authentication ־ **A**ניתן לאמת שכל משתמש הוא אכן מי שהוא אמור להיות

כדי לממש את הדרישות הללו, למערכות בסיסי נתונים רלציונים יש שפת Access Control Language עשירה המאפשרת להגדיר את ההרשאות הניתנות לכל משתמש בבסיס הנתונים ולהגדיר מודל הרשאות ברמת פעולות והרשאות. לדוגמה, ניתן להגדיר שלמשתמש מסוים יש הרשאה לבצע פעולת הוספה (insert) על טבלה מסוימת, ואם משתמש זה ינסה לבצע כל פעולה אחרת, הפעולה תכשל. דבר זה מתאפשר מכיוון שבבסיסי נתונים רלציונים למסד הנתונים יש מידע מדויק על מבנה הטבלאות ומסד הנתונים אוכף את מבנה הטבלאות באדיקות. במסד הנתונים הרלציוני של חברת Oracle יש אפשרות להגדיר הרשאות ברמת גרעיניות עוד יותר נמוכה. בתכונת Virtual Private Database הקיימת במסד הנתונים מגרסה 8i יש אפשרות להגדיר למסד הנתונים פונקציות המשנות באופן דינמי כל שאילתה המועברת למסד הנתונים

כך שלמנהל מסד הנתונים יש אפשרות להגדיר מדיניות אבטחה ברמת כל שורה בכל טבלה ומבט במסד הנתונים. לדוגמה, אם ביחס מסוים קיים טור המגדיר את רמת האבטחה של רשומה מסוימת, יכול מנהל מסד הנתונים להגדיר פונקציה הממפה משתמש מסוים לרמת אבטחה מסוימת, ומסד הנתונים ישנה כל שאילתה הנוגעת ביחס הזה כך שיתווסף תנאי לשאילתה באופן דינמי והתנאי הנדרש יבדק. הייחוד של תכונה זאת היא שלא ניתן לעקוף את הבדיקה, מכיוון שהבדיקה מבוצעת ברמת מסד הנתונים, ומדיניות האבטחה מחוברת לכל יחס ומבט במבט הנתונים שמנהל מסד הנתונים הגדיר [13]. בנוסף לכל זאת, מסדי נתונים רלציונים רבים מגינים על המידע גם ברמת קבצי המידע עצמם, ומאפשרים למנהל מסד הנתונים לשמור את קבצי המידע מוצפנים, בדר"כ במספר רמות - מרמת קבצים שלמים ועד רמת טורים מסוימים בטבלה (למשל ב־PostgreSQL [14]). מסדי נתונים רלציונים מגנים גם על פרוטוקולי הרשת שבהם משתמשים כדי להתחבר אל מסד הנתונים באמצעות הצפנת התעבורה, וחיבור מנגנון המשתמשים של מסד הנתונים אל שרתי הרשאות חיצוניים, דוגמת מערכת ההפעלה עליה מותקן מסד הנתונים, או שירותים כמו Kerberos או Active Directory מאפשר למסד הנתונים לאמת את זהות המשתמשים בו.

בחלק גדול מהמקרים לא ניתנה מחשבה מרובה לאבטחת נתונים במסדי הנתונים הלא רלציונים [15, 16, 17]. בגלל שבמסדי נתונים לא רלציונים לא תמיד יש למסד הנתונים מידע מוקדם לגבי מבנה הנתונים השמורים בתוכו, קשה יותר במסדי נתונים מסוג זה להגדיר ולקיים מודל הרשאות על הנתונים. למרות האופי המבוזר של מסדי נתונים מסוג NoSQL אין למסדי נתונים אלה בדר"כ מודעות לאבטחת הנתונים בין החלקים השונים של מסדי הנתונים המבוזרים, או אפילו על פרוטוקולי הרשת המשמשים לחיבור משתמש אל מסד הנתונים. ברוב המקרים אין במסדי נתונים אלה גם יכולת להצפין את המידע הנשמר בהם, ומסדי נתונים אלה חשופים להתקפות המבוצעות ברמת המחשב עליו הם מותקנים. בבדיקת יכולות האבטחה על מסדי הנתונים MongoDB ו־Cassandra נמצאו ליקויים רבים המאפשרים לתוקף לעקוף את מנגנוני הבקרה של מסד הנתונים ולהוציא מידע ממסדי הנתונים ללא הרשאה, או לפגוע בזמינות מסד הנתונים [18].

נכון להיום, מסדי נתונים לא רלציונים משמשים את הארגונים הגדולים ביותר, דוגמת Google, המשתמשים במסד נתונים בשם BigTable כבסיס למספר גדול של המוצרים שלהם [19], או Facebook המשתמשת במסד נתונים לא רלציוני כדי לממש את תכונת התקשורת המידית שלה [20]. בגלל היכולת של מסדי נתונים אלה להגיע לגדלים גדולים מאוד, מהר מאוד, יכולות האבטחה של מסדי נתונים אלה הופכת לחשובה יותר ככל שיותר חברות משתמשות במוצרים אלה כדי לבנות מוצרים חדשים.

בעבודה זאת אני סוקר שלושה מסדי נתונים לא רלציונים (Cassandra, HBase ו־MongoDB) על סמך מאמרים ועל סמך ניסיון אישי בהתקנת המוצרים וחקירתם. על חלק מבסיסי הנתונים הופיע מאמר Security Issues in NoSQL

Databases [18] שפורסם ב־TrustCOM 2011 (שהייתי המחבר הראשי שלו) שחקר את יכולות האבטחה של Cassandra ואת MongoDB. עבודה זו מרחיבה את הנאמר במאמר שפורסם בכך שמוסיף התייחסות ל־HBase, ומעדכן את תוכנו לגרסאות חדשות יותר של מסדי הנתונים הנסקרים שהופיעו במאמר שפורסם.

2 מבוא

כדי שניתן יהיה לסקור את תכונות האבטחה של מסדי נתונים לא רלציונים, צריך להציג קודם כל את צורת העבודה של מסד הנתונים, ולזהות את הנקודות החשובות שעליהם יש להגן בכל אחד ממסדי הנתונים. במסמך זה אתמקד בשלושה מסדי נתונים לא רלציונים הנמצאים בשימוש היום. תחילה אציג את מודל הנתונים של כל אחד מהם, את הארכיטקטורה של מסד הנתונים ואסקור בקצרה את ממשקי הגישה אל מסדי הנתונים האלה. אח"כ אגדיר רשימת תכונות שאותם רוצים לראות כשסוקרים את אבטחת מסדי הנתונים הללו, ואעבור על שלושת מסדי הנתונים הללו כדי להדגים כיצד מסדי נתונים אלה מקיימים (או לא) את הנדרש. אשווה גם מסד נתונים רלציוני לשלושת מסדי הנתונים הלא רלציונים ברשימת תכונות אלה, ואדון בהבדלים הנובעים מהארכיטקטורה המאפשרים למסד הנתונים הרלציוני להתנהג בצורה שונה מאלה שאינם רלציונים.

עבודה זאת מרחיבה את הנעשה ב־Security Issues in NoSQL Databases [18] שפורסם ב־TrustCOM 2011 ע"י עדכון המידע לגרסאות חדשות יותר של מסדי הנתונים הנסקרים (יחסית למאמר), והוספת מסד נתונים נוסף (HBase). בנוסף, במסגרת עבודה זאת מימשתי יכולות אימות מבוסס Kerberos ומנגנון הרשאות בבסיס הנתונים Cassandra הפותרים את הבעיות שהוצגו עבור מסד נתונים זה ב־Security Issues in NoSQL Databases. במהלך ביצוע העבודה, התקנתי את כל מסדי הנתונים הנסקרים בתצורתם המבוזרת, תוך הפעלת מנגנוני האבטחה הקיימים בהם.

בפרק 3.1 אסקור את מודל הנתונים מבוסס העמודות המשמש את Cassandra (גרסה 1.1.6) ואת HBase (גרסה 0.94.1-security) מעל Hadoop (גרסה 1.0.3), ואסקור את הארכיטקטורה של מסדי נתונים אלה. בפרק 3.2 אדון במודל הנתונים מבוסס מסמכים המשמש את MongoDB (גרסה 2.2.0) ואסקור את הארכיטקטורה של MongoDB. בפרק 3.3 אסקור שיטות שונות בהם ניתן לשלוף נתונים ממסדי הנתונים הנסקרים, בהתאם לממשקים שאלה מספקים. בפרק 3.4 אגדיר את רשימת התכונות אותם אבדוק כדי לאמוד את יכולות האבטחה של כל מסד נתונים. בפרק 4 אציג את הכלים והשיטות לביצוע של סדרת הבדיקות שאבצע כדי לסקור את יכולות האבטחה של מסדי הנתונים הנבדקים.

3 סקירת ספרות

3.1 מודל נתונים מבוסס עמודות

מסדי הנתונים Cassandra ו־HBase משתמשים במודל נתונים מבוסס עמודות שעקרונותיו מבוססים בעיקר על המאמר Bigtable [19]. בשני מסדי הנתונים, יחידת המידע הבסיסית מורכבת משלשה של נתונים: שם, ערך וחתימת זמן.

```
hbase(main):010:0> create 'example','cf1', 'cf2'
0 row(s) in 2.1890 seconds
hbase(main):012:0> put 'example', 'row1',
                       'cf1:col1', 'first value'
0 row(s) in 0.3090 seconds
hbase(main):013:0> put 'example', 'row1',
                       'cf1:col2', 'second value'
0 row(s) in 0.0120 seconds
hbase(main):015:0> put 'example', 'row1',
                       'cf2:col1', 'yet another value'
0 row(s) in 0.0660 seconds
hbase(main):016:0> put 'example', 'row2',
                       'cf1:diff', 'unique column '
0 row(s) in 0.0590 seconds
hbase(main):017:0> scan 'example'
ROW       COLUMN+CELL
row1      column=cf1:col1, timestamp=1350336034382,
                           value=first value
row1      column=cf1:col2, timestamp=1350336042723,
                           value=second value
row1      column=cf2:col1, timestamp=1350336059692,
                           value=yet another value
row2      column=cf1:diff, timestamp=1350336080931,
                           value=unique column
2 row(s) in 0.0890 seconds
```

איור 1: טבלה במודל מבוסס עמודות

חתימת הזמן היא מספר בגודל של 64 סיביות, וערכה נמסר ע"י האפליקציה המעדכנת את הנתון במסד הנתונים. הנתונים הנשמרים בחלקי השם והערך הם ערכים בינריים וערכם איננו מוגבל ע"י מסד הנתונים בשום צורה.

בשני מסדי הנתונים, נקודת ההנחה היא שישנם תמיד אוספי נתונים שהאפליקציה צריכה לנהל במשותף. במסדי נתונים רלציונים, נתונים הקשורים יחדיו היו מגיעים למסד הנתונים כחלק מאותה הטבלה. במסדי הנתונים Cassandra ן־HBase, נתונים הקשורים לוגית יחדיו יאורגנו במבנה הנקרא משפחת טורים (Column Family). באופן פורמלי, משפחת טורים היא אוסף של תאים בסיסיים בעלי משמעות משותפת הנשמרים יחדיו. ניתן להציג את המבנה בצורה הבאה: $(\text{Table, Row Id, Family, Column, Timestamp}) \rightarrow value$.

במאמר המקורי, טבלה אחת במסד הנתונים יכלה להכיל מספר גדול של משפחות טורים, כאשר הכוונה הייתה לשמור עד עשרות משפחות טורים בטבלה, ומספר לא מוגבל של טורים בכל משפחת טורים. נשים לב, שאין הכרח להגדיר מראש איזה טורים נמצאים במשפחת הטורים, ואין הכרח

שבכל שורה בטבלה יופיעו כל הטורים בכל משפחה, או אפילו כל משפחות הטורים שבטבלה. להבדיל ממסדי נתונים רלציונים בהם אם הוגדר טור ולא ניתן לו ערך בשורה אז נשמר עבורו ערך ריק (NULL), במסדי הנתונים הללו אין עלות לערך ריק, מכיוון שהוא פשוט לא נשמר.

לכל שורה בכל טבלה יש מזהה הנקבע ע"י האפליקציה. אין הגבלה על ערך המזהה - זה יכול להיות כל דבר ממספר סידורי למחרוזת סיביות ארוכה, ובלבד שהמזהה יהיה ייחודי בטבלה. השורות עבור כל טבלה נשמרות ממוינות, בדומה לאינדקס על מפתח ראשי בטבלה במסד נתונים רלציוני. העובדה שהטורים ממוינים מאפשרת למסד הנתונים להחזיר את הערכים בסדר הממוין בעת קריאת הנתונים ולפצל באופן אוטומטי את השורות בין מחשבים הנמצאים באשכול. האפליקציות המשתמשות במסדי הנתונים האלה מנצלות מצידן את התכונה כאינדקס על ערכי השורה. מסדי הנתונים משתמשים בשדה חתימת הזמן עבור כל טור כדי לנהל גרסאות עבור ערך נתון בכל שורה - יתכן שישמרו עבור שורה מסוימת וטור מסוים יותר מאשר ערך אחד. דבר זה נעשה כדי לאפשר למסד הנתונים לשכפל מידע בין מספר מחשבים באשכול המחשבים, ועדיין להיות מסוגל לבחור את הערך החדש ביותר כאשר מתקבלות יותר מתוצאה אחת לפעולת חיפוש על מפתח אחד. למרות שבשני מסדי הנתונים כל גרסאות המידע זמינות לאפליקציה, מומלץ למפתחים שלא להסתמך על קיומן של ערכים ישנים, שכן מסד הנתונים מתחייב לשמור רק את הגרסה האחרונה, וערכים ישנים יותר יכולים להימחק בזמן garbage collection ללא אזהרה לאפליקציה. באיור 1 אפשר לראות דוגמה לטבלה שהוגדרה עם שתי משפחות טורים. בשורה הראשונה, עם מזהה השורה row1, יש במשפחת הטורים הראשונה שני ערכים שונים ובמשפחת הטורים השנייה רק ערך אחד. בשורה השנייה, עם מזהה השורה row2 יש רק טור אחד במשפחת הטורים הראשונה. שמות הטורים לא נקבעו בזמן הגדרת הטבלה - רק בזמן הוספת הערכים לטבלה.

כל שתואר עד כה משותף לשני מסדי הנתונים. כעת נסקור את ההבדלים בין מסדי הנתונים ונעבור על הארכיטקטורה הפנימית של כל אחד ממסדי הנתונים.

3.1.1 הארכיטקטורה של Cassandra [1, 2]

3.1.1.1 ארגון הנתונים

ב־Cassandra אין אפשרות להגדיר טבלה, והנתונים מוכנסים ישירות למשפחות טורים. המידע בכל משפחת טורים ממויין לפי המפתח של השורות במשפחת הטורים, כאשר ב־Cassandra ניתן לשלוט בסוג המיון שיעשה. מסד הנתונים תומך בכמה סוגי מיונים - מספרי, לקסיקוגרפי לפי ASCII או Unicode, ועוד. אם הוגדר סוג מיון המגביל את המפתח של השורות במשפחת טורים מסוימת, מסד הנתונים לא יאפשר להכניס מפתח שאיננו מוגדר היטב עבור המיון. למשל - אם הוגדר שלמשפחת טורים יש מיון מספרי, על כל המפתחות

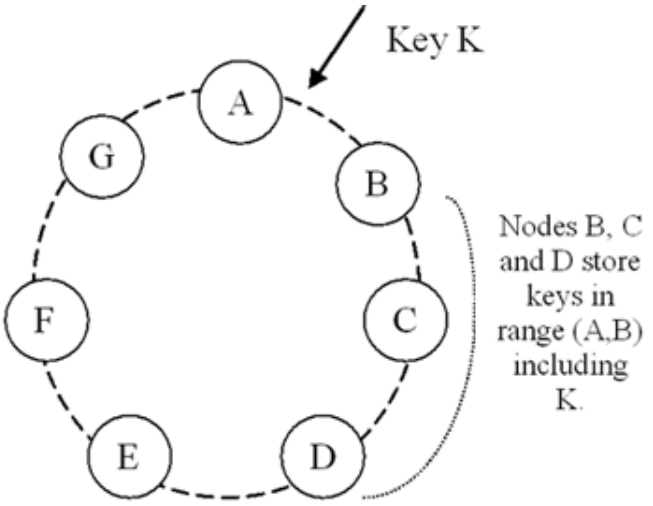
להיות מספריים, ומסד הנתונים לא יאפשר להכניס שורה בעלת מפתח שאיננו מספרי. משפחת טורים יכולה להכיל בעצמה עוד משפחות טורים. במקרה זה, נכנה את משפחת הטורים החיצונית בשם Super Column Family. תכונה זאת היא בעצם המקבילה של Cassandra לטבלאות המתוארות במאמר המקורי. ניתן לכנן רק רמה אחת של משפחות טורים ב־Cassandra. פעולת עדכון של שורה ב־Cassandra היא פעולה אטומית, ללא קשר למספר הטורים הנקראים או נכתבים בפעולה.

כאשר מגדירים את משפחת הטורים, ניתן (אך לא הכרחי) לספק מאפיינים עבור הטורים שישמרו במשפחת הטורים. בנוסף, ניתן להגדיר ערכי ברירת מחדל עבור טורים שעבורם לא הוגדר מידע מפורש. ערכי ברירת מחדל אלה ישמשו עבור כל טור שלא ניתנו עבורו הגדרות מפורשות. בנוסף, למשפחת טורים ישנם מאפיינים המגדירים כיצד יתנהג מסד הנתונים בשמירת ערכים לשורות של משפחת הטורים. בזמן השמירה, הטורים בכל שורה ממוינים בקובץ לפי שמותיהם. סוג המיון הוא מאפיין הניתן להגדרה בעת הגדרת משפחת הטורים. ניתן להגדיר מספר סוגי מיונים שונים עבור הטורים בשורה כדי לשנות את הסדר שבו טורים נשמרים בתוך שורה בודדת בקובץ של משפחת הטורים. Cassandra מספקת מספר סוגי מיונים שונים הכוללים מיון כמספר, מיון כמחרוזת Ascii, מיון כחתימת זמן, ועוד. ניתן גם לספק ל־Cassandra מימוש בשפת Java של פעולת השוואה מיוחדת בין שני ערכים, כדי להגדיר יחס סדר מיוחד עבור משפחת טורים.

משפחות טורים נשמרות בתוך מרחב מפתחות (Key Space). מרחבי מפתחות הן המקבילה של Cassandra לסכמות בעולם מסדי הנתונים הרלציונים, וזוהי הרחבה על המתואר במאמר המקורי (Bigtable). בהגדרת מרחב מפתחות, ניתן להגדיר ל־Cassandra פרמטרים השולטים בפיזור הנתונים של משפחות הטורים של מרחב המפתחות במחשבים המשתתפים במסד הנתונים.

3.1.1.2 ביזור שרתים

Cassandra הוא מסד נתונים מבוזר, המאפשר לחבר מספר גדול של מחשבים כצביר (אשכול) כדי לשפר את יכולת אפליקציה הבנויה מעל מסד נתונים זה להתמודד עם בעיות partitioning של הרשת ולשפר את זמני התגובה (availability) המורגשים ע"י משתמשים. יעדים אלה הוגדרו ע"י מפתחי מסד הנתונים כיותר חשובים מתכונת ה־consistency כפי שהוצגה בתאוריה של Eric Brewer. מסד הנתונים משתמש במפתחות של שורות בכל משפחת טורים על מנת לחשב את המחשב בצביר שלתוכו תשמר שורה מסוימת, באמצעות טכניקת consistent hashing [21] בצורה הדומה למימוש של מסד הנתונים Dynamo של חברת Amazon, המתואר במאמר Dynamo: Amazon's highly available keystore [8]. Cassandra מגבב את המפתח של השורה כדי למצוא את המחשב הנכון עבור השורה, ומכניס בנוסף את השורה לעוד מספר מחשבים נוספים (לפי מה שהוגדר לצביר). לדוגמה, באיור 2 ניתן לראות שעבור המפתח K, תוצאת הגיבוב היא אזור במרחב המפתחות אחרי שרת



איור 2: טבעת צביר ב־Cassandra. איור זה נלקח מ־[8]

A, ומכיוון שבצביר הזה הוגדר רמת שרידות של $n = 3$, הערך יוכנס לשרתים B,C ו־D. בעת השליפה, זהות המחשב הנכון ממנו צריך לשלוף את השורה מחושב באותה הצורה, והקריאה נעשית מהמחשב המתאים ־ במקרה זה שרת B. אם מחשב זה איננו זמין, יבדקו המחשבים האחרים אליהם הוכנסה השורה לפי סדר, כלומר קודם C ואח"כ D. כדי להחליט לאיזה מחשב בצביר תגיע הרשומה, מתייחסים לטווח פונקציית הגיבוב כתחום מעגלי (כלומר הערך הקטן ביותר עוקב אחרי הערך הגדול ביותר), ולכל מחשב בצביר מגרילים אזור שווה בגודלו בתחום. לאחר שמגבבים את ערך המפתח, מוצאים את המחשב הראשון שהמספר המייצג שלו גדול מהערך שחושב עבור מפתח השורה, ומחשב זה (ויתכן שמספר מחשבים אחריו במעגל) ישמרו את המידע. בצורה זאת, אם מחשב אחד איננו זמין, כל המידע שעליו נמצא גם במחשב שאחריו במעגל, ולא נאבד. ניתן להתאים את אסטרטגיית פיזור הנתונים עבור מרחב מפתחות מסוים כדי להתאים לתנאי טופולוגיה של רשת. לדוגמה, לחברה יש מספר מרכזי מידע באזורים גאוגרפיים שונים, והאפליקציה רוצה להבטיח שכל שורה תופיע בכל אחד ממרכזי המידע. דבר זה נעשה ע"י הקצאת מיקום על המעגל למחשבים מסוימים שלא בצורה אקראית אלא לפי אסטרטגיה מתאימה ־ לדוגמה, אם ישנם שני מרכזי מידע אז תמיד כל שני מחשבים עוקבים במעגל הגיבוב יהיו ממרכזי מידע שונים, ומספר המחשבים עליהם מחייבים כתיבה יהיה גדול מ־$n = 2$.

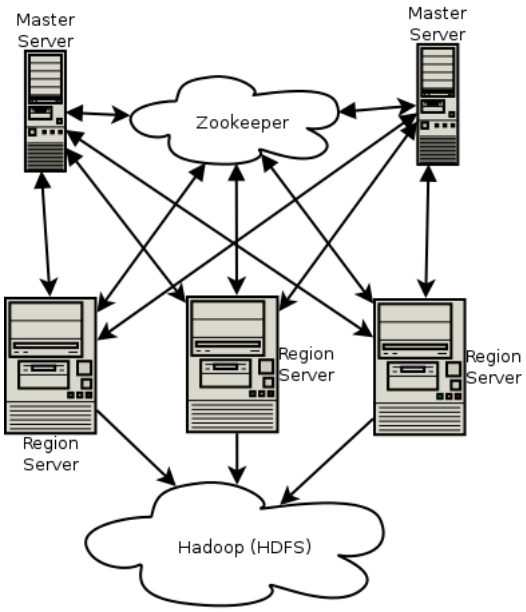
צריך להדגיש שבאשכול של Cassandra, אין שרת מרכזי או נקודה בודדת שהפלתה תגרום לכשל מיידי של כל האשכול.

3.1.2 הארכיטקטורה של HBase מעל Hadoop [3]

3.1.2.1 תשתיות בסיסיות ב־HBase

מודל הנתונים של HBase הוא זהה למתואר במאמר של BigTable שנסקר בסעיף 3.1. בעוד שמסד הנתונים Cassandra לא אימץ את אופן הביזור של המידע שתואר במאמר של BigTable אלא פנה למתודולוגיית ביזור אחרת, HBase נשאר הרבה יותר נאמן למאמר המקורי. כמו ש־BigTable משתמש במספר מוצרי תשתית כדי לאפשר ביזור נתונים, גם HBase משתמש במספר מוצרי תשתית כדי לאפשר לבנות צביר (אשכול) מחשבים גדול. תשתית ראשונה היא תשתית של מערכת קבצים מבוזרת בשם Hadoop [22]. תשתית זאת חושפת מערכת קבצים בה כל בלוק של נתונים מפוזר למספר מחשבים פיזיים שונים. מערכת הקבצים הזאת מיועדת לקבצים יחסית גדולים (קבצים הגדולים מ־64 מגה), אם כי גם שמירת קבצים קטנים יותר אפשרית. השימוש בתשתית זאת מבטיחה שלאחר ששרת כלשהו בצביר כתב את המידע למערכת הקבצים, המידע יהיה זמין לכל השרתים בצביר. Hadoop מטפל בפרטים הטכניים של שכפול הבלוקים בין שרתים שונים, והבאת המידע מהדיסק המקומי שעליו הוא מאוחסן למקום בו המידע נדרש. Hadoop יכול לבחור בכל פעולת קריאה את השרת הכי פחות עמוס שעליו המידע זמין (בדר"כ יש יותר מאחד כזה) ולנתב אליו את פעולת הקריאה. Hadoop היא מימוש חופשי של מוצר תשתית פרטי של חברת Google בשם Google File System [23] (או GFS בקצרה).

תשתית שנייה עליה מתבסס HBase הוא תשתית בשם Zookeeper [24]. תשתית ה־Zookeeper מאפשרת לאשכול ה־HBase לתאם בין כל השרתים המהווים את האשכול את התפקיד הנוכחי של כל שרת באשכול. אשכול של HBase מכיל מספר סוגי שירותים (פירוט בהמשך) וכדי לאפשר לכל מי שדורש את המידע למצוא את השרת המתאים ביותר לפי סוג השירות הנדרש, האשכול מנהל את המידע הזה בצורה מבוזרת. אין הכוונה למידע הנשמר במסד הנתונים, אלא למידע המשמש לניהול של האשכול. אחד היתרונות לשיטה זאת היא שלא נדרש לממש פרוטוקול Heartbeat בין כל שרת באשכול (לדוגמה ע"י שליחת ping) כדי לוודא עבור כל שרת ששאר השרתים באשכול זמינים, מכיוון שברגע ששרת באשכול נעלם, תשתית ה־Zookeeper דואגת ליידע את כל שאר האשכול. משמעות הדבר שאירוע מסוג זה מתגלה לשאר חברי האשכול באותו רגע בו קרה, ולא בזמן שליחת הודעת ה־Heartbeat הבאה. תשתית ה־Zookeeper עצמה מנוהל כאשכול מחשבים, ומידע הנשמר בתוך Zookeeper עבור HBase משוכפל בין חברי אשכול Zookeeper ברגע שהוא מתעדכן. התיעוד של Zookeeper ממליץ על אשכול המכיל מספר אי־זוגי של שרתים, ולפחות שלושה חברים באשכול. אשכול ה־Zookeeper זמין כל עוד יותר ממחצית מחבריו זמינים, ולכן השרידות שהוא מספק תלויה במספר השרתים שהוקצו לאשכול.



איור 3: ארכיטקטורת HBase

3.1.2.2 ארכיטקטורת מסד הנתונים HBase

כפי שניתן לראות באיור 3, קיימים באשכול שני סוגים של שרתים. שרתי Region Server אחראיים על שמירת נתונים בתוך מערכת הקבצים המבוזרת, ומנהלים את התקשורת עם אשכול ה־Hadoop. בכל אשכול יש שרת אחד המשמש שרת ראשי (Master Server), והוא אחראי על פיזור האחריות על הנתונים הנשמרים באשכול כולו. למטרות שרידות, ניתן להגדיר מספר מכונות המריצות את רכיב התוכנה של ה־Master Server, אבל רק שרת אחד בפועל עושה את העבודה בכל רגע נתון. הסנכרון בין שרתים אלה נעשה ע"י שמירת נעילה על משאב אחד באשכול ה־Zookeeper המשמש את אשכול מסד הנתונים. כל השורות בכל טבלה המוגדרת במערכת מחולקים לאזורים (Regions) לפי מפתח השורה. השרת הראשי של האשכול אחראי לחלק את תחומי המפתחות הקיימים במערכת בין שרתי Region Server הזמינים, כך שבכל רגע נתון יש רק שרת אזורי אחד האחראי על קריאה או כתיבה של אזור אחד. נשים לב שבסופו של דבר הנתונים עצמם נשמרים למערכת הקבצים המבוזרת ולכן זמינים לכל השרתים באשכול - ולכן השרת הראשי יכול לשנות את הגדרות האזורים. לדוגמה, אם אזור אחד מתמלא כך שהוא מכיל יותר מדי מידע, השרת הראשי יפצל את האזור לשני אזורים, ויעביר את האחריות על אחד האזורים לשרת אחר. אם שרת האזורים כזה קורס, השרת הראשי מעביר את האזורים שנוהלו ע"י השרת שקרס לשרת אזורים אחר הקיים באשכול. תשתית ה־Zookeeper מכילה הפניה לאחד משרתי האזורים המכיל אזור מערכת מיוחד, המתאר את חלוקת האזורים לכל אחד משרתי

האזורים. מידע זה נשמר בתוך Hadoop כטבלת מערכת, ומשמש למציאת השרת המתאים ביותר לביצוע פעולות קריאה/כתיבה.

3.2 MongoDB [4] - מסד נתונים מבוסס מסמכים

לפני שניתן להציג את מודל הנתונים של MongoDB נציג תחילה שפה הנקראת BSON [25] (Binary JSON). נגדיר אוביקט בסוני באוסף של תכונות, ונגדיר תכונה כצירוף של שם וערך $(name, value)$. השם חייב להיות מחרוזת קריאה לפי תקן Unicode, בעוד הערך יכול להיות או פריט מידע בסיסי (למשל מחרוזת, מספר, תאריך, ערך בוליאני או ערך בינרי), או פריט מידע מרוכב (רשימה של ערכים, אוביקט מרוכב או רשימה של אוביקטים מורכבים). מסמך BSON ניתן להמרה מיידית לפורמט טקסטואלי בשם JSON (JavaScript Object Notation), המאפשר להציג מסמכים לעין האנושית. מסמך JSON מיוחד בכך שייצוגו הוא גם חוקי בשפת התכנות JavaScript, דבר המאפשר לשמור ולטפל במסמכים מסוג זה בתוך תוכניות. ראה איור 4 לדוגמה של מסמך JSON והמקבילה ה-XML-ית שלו. התקן ה-XML-י מאפשר למשתמש תכונות רבות - מהגדרת סכמה וביצוע ווידוא לתוכן של מסמך, שפות שאילתה על מסמכים ועד יצירת קשרים לוגיים בין מסמכים שונים. מסיבה זאת, פורמט ה-XML נחשב מסובך מדי ע"י מפתחים רבים - תמיכה מלאה ב-XML הצטיירה כהרבה עבודה על יכולות שלא הזדקקו להם במסד הנתונים שנבנה, והממשק למפתח האפליקציה (API) של שפת XML נחשבה למורכבת יחסית לנדרש מהמפתח במסמכי JSON. מסיבה זאת, MongoDB איננו משתמש בפורמט XML אלא בפורמט BSON שנתפס ע"י המפתחים כבעל כל היתרונות של JSON ובנוסף גם קל לשימוש למטרות הזרמה (Streaming).

MongoDB הוא מסד נתונים השומר נתונים באוספים של מסמכים בפורמט BSON כאשר המגבלה היחידה ש-MongoDB מציב לנתונים הוא שבכל מסמך הנשמר במסד הנתונים חייבת להיות קיימת תכונה מחרוזתית בשם _id המשמשת כמפתח המזהה את המסמך. המסמכים מאוגדים מטעמי נוחות באוספים, ומסד הנתונים איננו מחייב הגדרה מראש של האוספים האלה ואיננו מגביל את מספר האוספים שניתן להגדיר. המסמכים הנשמרים בתוך כל אוסף יזוהו באופן חד-חד ערכי על-ידי התכונה _id. מסד הנתונים איננו מחייב שהמסמכים הנמצאים בתוך אוסף יהיו דומים אחד לשני. שני מסמכים הנמצאים בתוך אותו האוסף יכולים להיות בעלי תכונות אחרות לגמרי.

יחידת העבודה הבסיסית ב-MongoDB היא מסמך אחד שלם. כדי לבצע פעולה ב-MongoDB, צריך קודם כל לזהות למסד הנתונים את המסמך שאותו רוצים לעדכן, ואח"כ את אוסף השינויים שרוצים לבצע בו. לא ניתן לבצע סדרה של שינויים במספר מסמכים בפעולה אטומית אחת במסד הנתונים. פעולת עדכון במסמך בודד יכולה לעדכן כל חלק מהמסמך, מתכונה אחת ועד אוסף כל התכונות במסמך.

```xml
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

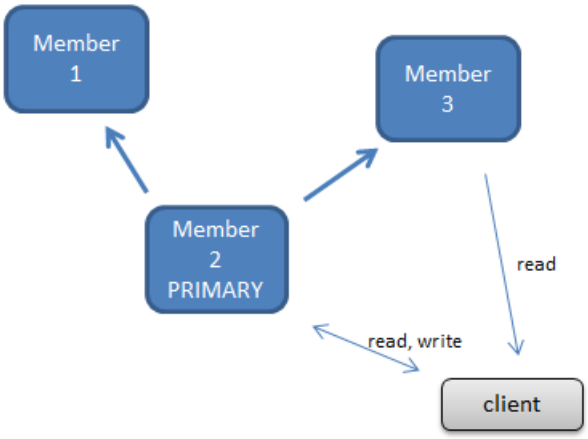
```json
{
"menu": { "id": "file", "value": "File",
  "popup": { "menuitem": [
              {"value": "New", "onclick": "CreateNewDoc()"},
              {"value": "Open", "onclick": "OpenDoc()"},
              {"value": "Close", "onclick": "CloseDoc()"}
                          ]
           }
        }
}
```

איור 4: מסמך JSON ומסמך XML

3.2.1 הארכיטקטורה של MongoDB

3.2.1.1 צביר שכפול

במתכונתו הבסיסית ביותר, MongoDB מורץ כתהליך אחד על גבי מכונה אחת. במתכונת זאת, מסד הנתונים איננו מבוזר ואין לו יכולת להתמודד עם עומס גדול במיוחד או לספק יתירות גבוהה. כדי לאפשר למסד הנתונים להתמודד עם בעיות חומרה ולספק יתירות גבוהה, נוספה תמיכה ב־MongoDB בשכפול כל הנתונים הנמצאים בתוך מסד הנתונים בצביר. יכולת זאת נקראת Replica Set והיא מאפשרת למסד הנתונים לשכפל באופן אוטומטי את כל הנתונים בין עד 12 מחשבים בצביר. המימוש של תכונה זאת מאפשר לכל חברי הצביר לזהות כשל בכל אחד מהחברים בצביר, ובמקרה וזוהה כשל, מסד הנתונים מתמודד עם הכשל באופן אוטומטי - ללא התערבות מפעיל. בתוך הצביר ישנו מחשב אחד הנחשב ראשי (Primary) ושאר המחשבים נחשבים משניים (Secondary). תפקיד המחשב הראשי ניתן לאחד מהמחשבים בצביר אחרי תהליך הצבעה בין כל החברים בצביר. ניתן לבצע פעולות קריאה מכל אחד מהמחשבים בצביר (ראשיים ומשניים), אבל פעולות עדכון ניתן לבצע רק על המחשב הראשי. במידה והמחשב הראשי נופל, יבחר אחד מהמשניים לתפקיד המחשב הראשי, ואפליקציות העובדות עם הצביר יכולות להמשיך את עבודתם. כאשר המחשב שנפל יתאושש ויתחבר מחדש לצביר, הוא יהפוך למחשב משני. פעולות קריאה המבוצעות כנגד מחשב ראשי יהיו תמיד מעודכנות לחלוטין - כלומר יחזירו את הגרסה האחרונה של כל המסמכים במסד הנתונים. פעולות קריאה המבוצעות כנגד מחשב משני



איור 5: צביר שכפול ב־MongoDB איור נלקח מ־[4]

יכולות להחזיר גרסה ישנה יותר של הנתונים יחסית למידע הקיים במחשב הראשי, ועל האפליקציה לדעת להתמודד עם עניין זה. תכונה זאת הופכת את MongoDB בתצורה זאת למסד נתונים הנחשב Eventually Consistent. תכונת השכפול ממומשת ע"י שימוש בקובץ תנועות (Operation Log). כל פעולת עדכון המבוצעת בשרת הראשי נכתבת לקובץ התנועות ולקובץ הנתונים שבמחשב הראשי. קובץ התנועות משוכפל אל כל המחשבים המשניים באופן אסינכרוני וכל המחשבי המשניים מבצעים את כל הפעולות הכתובות בקובץ התנועות. באיור 5 ניתן לראות צביר המורכב משלושה מחשבים, ולקוח אחד העובד מול הצביר. הלקוח ניגש למחשב הראשי לביצוע פעולות כתיבה, וכמו כן הוא ניגש לראשי ולמשני במקביל כדי לבצע פעולות קריאה, ובצורה זאת העומס מפוזר בין כל חברי הצביר.

מכיוון שהצביר מחליט מיהו המחשב הראשי ע"י שימוש בהצבעה, מומלץ מאוד שבצביר יהיה מספר אי־זוגי של מחשבים. כדי לאפשר מספר אי־זוגי של מחשבים בצביר במידה ולא ניתן להעמיד מספיק חומרה, קיים סוג נוסף של חבר בצביר הנקרא Arbiter. מחשב בעל תפקיד זה יכול להשתתף בהצבעות, אבל איננו מכיל מידע ולא יכול להפוך בעצמו למחשב ראשי בצביר. ניתן להוסיף מחשב מסוג זה לצביר כדי שישמש שובר שוויון וכדי לוודא שתמיד יש מספר אי־זוגי של מחשבים בצביר, ומובטח שהעלות התפעולית (כמות משאבי מחשוב) הנדרשים עבור מחשב עם תפקיד זה היא מינימלית.

ניתן לשלוט עד רמה מסוימת בתהליך ההצבעה ע"י נתינת משקל לכל מחשב בצביר שיכול לשפר את סיכויו של המחשב להפוך למחשב ראשי בהצבעה. תכונה זאת מאפשרת, לדוגמה, למקם חברים בצביר ביותר ממרכז מחשוב אחד המופרדים גאוגרפית, ולתת עדיפות לכל המחשבים בצביר הנמצאים באתר אחד (הראשי) על כל המחשבים הנמצאים באתרים אחרים.

בנוסף, ניתן להגדיר חבר מוסתר בצביר שכפול. מחשב מוסתר הוא מחשב שלא מפרסם את נוכחותו בצביר ולכן ניתן להתחבר אליו רק אם כתובתו ידועה מראש. מחשב כזה לעולם לא יהפוך להיות מחשב ראשי בצביר. שימושים אפשריים לחברים מוסתרים הם גיבוי של המידע בצביר, או מכונה ייעודית להרצת דוחות.

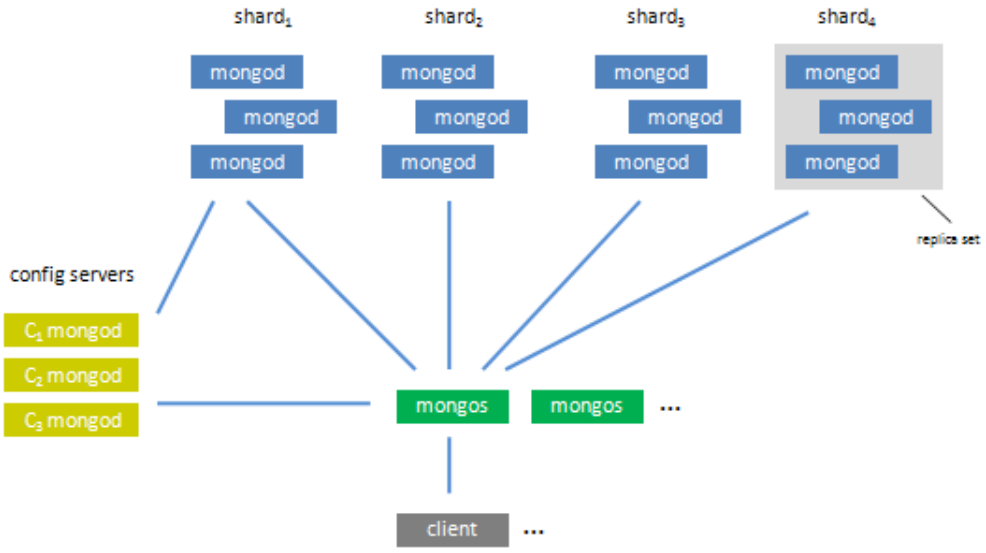
3.2.1.2 צביר פיזור (Sharding)

החיסרון של צביר שכפול היא שפעולות עדכון יכולות להתבצע רק כנגד המחשב הראשי בצביר, ולכן פתרון זה איננו מקנה למסד הנתונים לעבוד ברמת ביצועים גבוהה כאשר האפליקציה מבצעת פעולות עדכון רבות. כדי להקל על הבעיה, נוספה למסד הנתונים יכולת נוספת של פיזור הנתונים בין מספר צבירי שכפול. כדי לתמוך במתכונת זאת, צריך להגדיר במסד הנתונים מפתח שערכיו משמשים לפיצול הנתונים בין מספר מחשבים ראשיים, תוך יצירת מספר צבירים של מסד הנתונים שבכל אחד מהם יש רק מסמכים המכילים ערכים מסוימים במפתח הפיצול שהוגדר. יכולת זאת נקראת באנגלית Sharding. שימוש ב־Sharding איננו ייחודי ל־MongoDB. לדוגמה, גם Cassandra ו־Hadoop מפזרים את הנתונים הנמצאים בתוכם בין חברים בצביר עפ"י מפתח. הייחוד של MongoDB הוא בכך שהפיזור נעשה ברמת מסמך שלם ולא ברמת שורה בודדת בטבלה, וש־MongoDB דואג לפזר את הנתונים בצורה שווה בין כל החברים בצביר באופן אוטומטי, ללא צורך בהתערבותו של מפעיל. באיור 6 ניתן לראות דוגמה לצביר המנצל גם את תכונות השכפול של מסד הנתונים וגם את תכונות הפיזור של מסד הנתונים.

כפי שניתן לראות באיור 6, כל אחד מהחלקים Shards בצביר יכול בעצמו להיות צביר שכפול כדי לוודא שיש יותר מהעתק אחד של הנתונים. קיימים שני סוגי ישויות נוספים בצביר מסוג זה. סוג ראשון הוא שרתי קונפיגורציה. שרתים אלה מכילים את המידע הנדרש כדי למצוא באיזה מקום בצביר נמצא כל פריט מידע בכל רגע נתון. ישות שנייה המסומנת באיור 6 בצבע ירוק היא ישות האחראית לנתב בקשות של אפליקציות הניגשות אל מסד הנתונים אל החלק הנכון בצביר, ולהחזיר אל האפליקציה את התשובה. ישות זאת היא בעצם הישות מולה עובדות האפליקציות ־ האפליקציה איננה צריכה לנהל בעצמה את הפניות לכל אחד מהחלקים של הצביר כדי להגיע לנתונים שהיא צריכה. הצביר עצמו מנהל את הנתונים שבתוכו, ואם הצביר מזהה שחלק מסוים מכיל יותר מידע יחסית לחלקים אחרים, המידע יועבר לחלקים האחרים באופן אוטומטי כך שהעומס יפוזר בין כל המחשבים בצביר.

3.3 גישה לנתונים במודל מבוזר

שלושת מסדי הנתונים הנסקרים אינם תומכים בשפת SQL כדי לגשת למידע. תמיכה מלאה בשפת SQL הוגדרה ע"י מפתחי מסדי הנתונים האלה כקשה למימוש ופחות מתאימה למודל הנתונים שמסדי הנתונים חושפים. עבור



איור 6: צביר MongoDB מורכב. איור נלקח מ־[4]

MongoDB שפת SQL איננה מתאימה מכיוון שמודל הנתונים ש־MongoDB מכתיב הוא מבוסס מסמכים ולא רשומות, ועבור Cassandra ו־HBase שפת השאילתות הסטנדרטית איננה מתאימה מכיוון שמסדי הנתונים האלה אינם מחייבים הגדרת סכמה באופן מסודר, ואין הכרח שבכל שורה כאותה הטבלה יהיו אותם הטורים. בנוסף, אין מסדי הנתונים האלה תומכים בפעולות Join בין טבלאות הנשמרות בהן.

לפני שאסקור את אפשרויות הגישה למסדי הנתונים הנסקרים, אציג מספר טכנולוגיות המשמשות כתשתית לגישה למסדי נתונים אלה.

3.3.1 Thrift [5]

מוצר זה נולד בחברת Facebook לאחר שהמפתחים בחברה נדרשו לפתח תשתית רשתית לכתיבת שירותי Remote Procedure Call (RPC) שתהיה מסוגלת לעבוד בצורה יעילה במספר גדול של שפות תכנות. התשתית של חברת Facebook מורכבת ממספר גדול של שירותי רשת, כשכל שירות נכתב בשפה שצוות הפיתוח האחראי עליו חשב שיספק את הביצועים הכי טובים. הבעיה של המפתחים הייתה שהיה צורך לאפשר גישה בין אוספי השירותים השונים, בצורה יעילה וקלה, וללא צורך לפתח מחדש את כל התשתית בכל פעם ששירותים כאלה נוספו או התעדכנו. הפתרון שנבחר היה ליצור הפרדה בין שני חלקים לוגיים:

1. שירותי רשת תשתיתיים - כל הקוד הנדרש כדי לבצע את החלק הטכני

```
namespace cpp Example
namespace java name.okman.example

struct DataStructure {
        1: i32 number,
        2: string text,
        3: list<binary> binaryData,
}
service DoSomething {
        void handleData(1:DataStructure data)
}
```

איור 7: ממשק Thrift לדוגמה

בתקשורת בין תוכניות

2. ייצוג המידע המועבר - היכולת של כל פרויקט להגדיר את המידע אותו הפרויקט מוכן לקבל ואת המידע אותו הפרויקט מחזיר

ספריית Thrift מגדירה שפה לתיאור ממשקים בשם Thrift IDL (Thrift Interface Definition Language) המאפשרת לרכיב תוכנה אחד להגדיר בצורה שאיננה תלויה בשפה מסוימת את מבנה הנתונים שיועברו. יחד עם הספרייה מסופק מהדר היודע לתרגם את הקלט משפת Thrift IDL לשפת התכנות המיועדת. בגרסה הנוכחית (0.8.0) יש תמיכה בשפות C++, C#, Cocoa, D, Delphi, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby ו-Smalltalk. התוצר המתקבל מהפעלת המהדר בכל אחד מהשפות הוא מבנה נתונים המאפשר למפתח לייצג בצורה הטבעית לשפה הודעה מסוימת, ובאמצעות ספריית Thrift ניתן לשלוח ולקבל הודעות כאלה ללא שהמפתח נדרש לכתוב כמות גדולה של קוד. הספרייה מאפשרת למפתח להגדיר פרוטוקול ע"י הגדרת אוסף ההודעות המרכיבות את הפרוטוקול. הספרייה מספקת את כל הנדרש כדי לשלוח ולקבל הודעות בפרוטוקול, ומשאירה למפתח להוסיף את הלוגיקה הנדרשת כדי לטפל בהודעות נכנסות וכדי להרכיב את ההודעות היוצאות.

באיור 7 ניתן לראות דוגמה לממשק שהוגדר בThrift IDL המגדיר מבנה נתונים בשם DataStructure ושירות בשם DoSomething המגדיר הודעת פרוטוקול בודדת בשם handleData. באמצעות המהדר של Thrift ניתן לייצר באופן אוטומטי את כל הקוד הנדרש לתוכניות בכל השפות הנתמכות כדי להפעיל את השירות DoSomething. הספרייה תטפל ברמת התקשורת, וכל שיידרש מהמפתח זה לממש את הפונקציה handleData.

3.3.2 [6] AVRO

ספריית AVRO מספקת שירות דומה לספריית Thrift, עם מספר הבדלים קטנים. כאשר משתמשים ב־AVRO, אין הכרח להדר מראש את קבצי ההגדרה של הפרוטוקול לקוד בשפת היעד. ההודעה המועברת מכילה בתוכה את כל ההגדרות הנדרשות כדי שהצד המקבל יוכל לטפל בצורה נכונה בהודעה. ספריית AVRO משתמשת בקבצי הגדרה הכתובים בשפת JSON כדי להגדיר את מבני הנתונים, ההודעות והפרוטוקולים הנדרשים. ניתן (אך לא נדרש) לכתוב את כל אלה גם בשפת IDL ולהשתמש במהדר כדי לקבל את קובץ ההגדרות בצורת JSON. השיפור המשמעותי של ספריית AVRO הוא בכך שניתן לייצר את הסכמה של מבני הנתונים המשודרים באופן דינמי. דבר זה מתאים לתפיסה שלא קיימת הגדרה מסודרת של המידע המשודר לפני שהוא משודר.

ספריית AVRO נוצרה כחלק מפרויקט Hadoop בין השאר כדי לאפשר לשרתים ולקוחות של צביר Hadoop בגרסאות שונות לעבוד ביחד. לפני השימוש בספרייה, נדרש שכל הלקוחות של צביר Hadoop ושכל השרתים המשתתפים בצביר יריצו את אותה הגרסה של התוכנה כדי לוודא שהפרוטוקול יהיה אותו הפרוטוקול.

3.3.3 [7] Representational State Transfer

ארכיטקטורת REST היא ארכיטקטורה לבניית שירותים מבוזרים בצורה המקלה על השרת ומעמיסה אחריות על הלקוח. כדי לתאר את הארכיטקטורה, נגדיר תחילה את אבני הבניה הנדרשים:

משאב משאב (Resource) הוא כל יחידת מידע אחת בודדת הניתנת לייצוג בצד השרת אשר יש לו שם יחודי. דוגמאות למשאבים הם דף באתר באינטרנט, טבלה במסד נתונים, משתמש במערכת מבוזרת, ועוד.

מזהה־משאב מזהה־משאב (Resource Identifier) הוא מחרוזת המזהה באופן חד־חד ערכי משאב אחד מסוים בנקודת זמן מסויימת. מזהי משאבים באינטרנט הנם מחרוזות URI (כפי שמתואר ב־RFC 2396): [scheme://][authority]path[?query]. לדוגמה, מזהה משאב המתאר את אתר הבית של מנוע החיפוש Google יכתב כך: http://www.google.com/ כאשר הסכמה כאן היא http, ה־authority הוא כתובת האינטרנט של השרת של Google , המסלול הוא המסלול הריק ולא נעשה שימוש ברכיב השאילתה.

באופן מעט יותר מדויק, משאב R הוא יחס השיוך $M_R(t)$ הממפה בין מזהיי משאב כפונקציה של זמן למשאב כלשהו. יתכן שמזהה משאב מסוים יעבוד רק לפרק זמן קצר, או שלא ישתנה בכלל כפונקציה של זמן. יתכן בהחלט שיהיה ניתן להגיע לאותו המשאב דרך יותר ממזהה משאב אחד. לדוגמה, מזהה המשאב "מעבד המחשב של מחבר המאמר" ומזהה המשאב "Intel Core-i7" יכולים להצביע לפרק זמן מסוים על אותו המשאב, אבל לאחר שמחבר

המאמר יחליף את מעבד המחשב שלו, המזהה הראשון כבר לא יצביע על אותו המשאב כמו המזהה השני. שירותים בארכיטקטורת REST יכולים לעשות שימוש בהגדרה הזאת כדי ליצור מספר גדול (או קטן, כנדרש) של מזהים עבור אותו המשאב.

ארכיטקטורת REST משתמשת במזהה משאב כדי לציין לשרת את המשאב שעליו נדרש השרת לבצע פעולה. כל בקשה בארכיטקטורת REST מורכבת משלוש חלקים:

1. המשאב עליו נדרשת פעולה
2. הפעולה הנדרשת לביצוע
3. המידע הרלוונטי לביצוע הפעולה

אסופת שלושת החלקים האלה נקראת ייצוג (Representation) הבקשה. לדוגמה, פעולת חיפוש במנוע החיפוש של Google ניתנת לתיאור ע"י מזהה המשאב http://www.google.com/#hl=en&q=REST+example. בהפעלת מזהה המשאב הזה דרך הדפדפן, תשלח בקשה לשרת החיפוש (www.google.com) בפרוטוקול המתואר בסכמה (http) כאמצעות פעולת GET המוגדרת בפרוטוקול http, והמידע הנוסף שמועבר מכיל את כל הנדרש לשרת כדי לבצע את פעולת החיפוש. במקרה זה, מזהה המשאב מבקש שהתוצאה תוחזר באנגלית (hl=en) ושיבוצע חיפוש על המילים "REST example". קל לראות ששרת החיפוש יכול לבצע את הפעולה הנדרשת ללא כל ידע מוקדם על הלקוח המבקש לבצע את הבקשה. הלקוח העביר אל השרת את כל המידע הנדרש לפעולה. בתשובת השרת, יתכן ויועבר ללקוח מזהה נוסף שיקל על השרת לבצע פעולת המשך (לדוגמה הבאת דף נוסף של תוצאות). בפרוטוקול http ישנם מספר פעולות נוספות המוגדרות בתקן, כגון PUT, POST ו־DELETE ואפשר להשתמש בהם כדי למפות אוסף פעולות עדכון, הוספה, מחיקה ושליפה למזהי משאבים. נמפה פעולת שליפה לפקודת GET בפרוטוקול http, פעולת מחיקה לפקודת DELETE, פעולת יצירה לפקודת PUT ופעולת עדכון לפקודת POST. באופן זה ניתן לממש שירותים מבוזרים הכוללים גישה למסדי נתונים באמצעות פרוטוקול http בארכיטקטורת REST.

בעוד שברמת התאוריה אין הכרח להשתמש דווקא בפרוטוקול HTTP כדי לעבוד בארכיטקטורת REST, הסמנטיקה של פרוטוקול HTTP נוחה מאוד לארכיטקטורת REST ולכן פרוטוקול HTTP משמש ברוב המקרים כפרוטוקול להעברת הודעות במערכות המשתמשות בארכיטקטורת REST. כמובן שאין מניעה להשתמש בעקרונות הארכיטקטורה מעל כל פרוטוקול תקשורת אחר. לדוגמה, פרוטוקול FTP יכול לשמש כפרוטוקול בסיס לארכיטקטורת REST: ניתן לתאר בו משאבים ע"י מסלול, ניתן לקודד את הפעולה הנדרשת על המשאבים לפי הפקודה הרלוונטית של פרוטוקול FTP (מחיקה, הוספה, שליפה של משאב), וניתן להעביר כל מידע הרלוונטי לביצוע פעולה כתוכן של קובץ המועבר לשרת ה־FTP.

3.3.4 גישה לנתונים ב־Cassandra

מסד הנתונים מאפשר כמה רמות של גישה אל הנתונים הנשמרים בתוכו. ברמה הבסיסית ביותר, ניתן להשתמש ברמת ה־API הפנימית של Cassandra המשמשת את מסד הנתונים לתקשורת בין חברי הצביר. רמה זאת נקראת StorageAPI, וזו הצורה המורכבת ביותר אבל היעילה ביותר האפשרית כדי לגשת אל הנתונים. ממשק זה כתוב בשפת Java ולכן זמין רק לתוכניות הכתובות בשפה זאת. בנוסף, מכיוון שממשק זה נועד לתקשורת בין החברים בצביר לבין עצמם, מסד הנתונים יראה אפליקציה המחוברת באמצעות ממשק זה כחבר מנוון בצביר. מסיבות אלה, השימוש בממשק זה איננו מומלץ לשימוש ע"י מפתחי הפרויקט, ואלה בנו מעל ממשק זה מספר ממשקים אחרים הנחשבים עדיפים.

מפתחי פרויקט Cassandra מספקים יחד עם קוד המקור של Cassandra קבצי הגדרה של ספריית Thrift המאפשרים ללקוחות העובדים עם ספרייה זאת לתקשר עם מסד הנתונים בכל שפה הנתמכת ע"י פרויקט Thrift. המימוש הפנימי של הודעות הפרוטוקול משתמש ב־StorageAPI כדי לבצע את העבודה, ומימוש זה מתוחזק ע"י הפרויקט כך שלקוח של מסד הנתונים איננו צריך להשתמש בעצמו ב־StorageAPI. מסיבה זאת, ממשק Thrift נחשב לממשק המומלץ לשימוש ע"י הפרויקט.

בגרסאות האחרונות של Cassandra יש מימוש של שפת שאילתות הנקראת CQL ־ (Cassandra Query Language). שפת שאילתות זאת חושפת את כל היכולות של Cassandra בצורה דומה ל־SQL תוך התחשבות בהבדלים בין מודל הנתונים במסד נתונים רלציוני למודל הנתונים הנמצא בשימוש ב־Cassandra. CQL מאפשר שליטה בחתימת הזמן המוצמדת לכל נתון במסד הנתונים, מאפשר שליטה ברמת האמינות (consistency) הנדרשת לביצוע פעולות ע"י ציון מספר הכותבים המינימלי הנדרש להצלחת הפעולה, ועוד. CQL ממומשת ע"י סט של נוסף של הודעות הממומשות מעל Thrift, כך שהשימוש בשפת השאילתות אפשרי בכל שפת תכנות הנתמכת ב־Thrift. חשוב להדגיש שלא קיימת פונקציונליות ב־CQL שאיננה קיימת ע"י שימוש בממשק ה־Thrift הרגיל. CQL חושפת את כל היכולות של Cassandra ע"י הודעת Thrift אחת בלבד (execute_cql_query), ולכן השימוש ב־CQL הוא פשוט יותר למפתח, ובכך החוזק של שפת השאילתות.

קיימת אינטגרציה בין פרויקט Cassandra ופרויקט Hadoop המאפשרת להריץ שאילתות MapReduce באמצעות תשתית Hadoop כך שמקור המידע הנבדק יהיה מסד נתונים מסוג Cassandra. בצורה זאת ניתן לעשות שימוש בתוכניות MapReduce מעל בסיס נתונים זה.

3.3.5 גישה לנתונים ב־HBase

הגישה היעילה ביותר ל־HBase מתבצעת באמצעות API הכתוב בשפת Java (השפה שבה השתמשו כדי לכתוב את HBase). ממשק זה הוא הבסיסי ביותר

וגם היעיל ביותר, אבל יש לו מגבלה אחת גדולה - הוא זמין רק לתוכניות שנכתבו בשפת Java. בדומה ל־Cassandra, גם HBase מספק מספר ממשקים המשתמשים בממשק הבסיסי כדי לאפשר גישה למסד הנתונים בשפות אחרות. מסד הנתונים מספק שלושה מוצרי תרגום (Application Gateways) המאפשרים גישה למסד הנתונים דרך ממשקי Thrift, AVRO ו־REST מעל פרוטוקול http. כדי להשתמש בממשקים האלה יש צורך להפעיל את מוצרי התרגום, ולאחר שאלו הופעלו ניתן לגשת את מסד הנתונים בכל שפה הנתמכת ע"י הפרוטוקולים האלה.

כאשר HBase מותקן מעל Hadoop, ניתן להשתמש גם בתשתית ה־MapReduce של פרויקט Hadoop כדי להפעיל שאילתות באמצעות אלגוריתם MapReduce על נתונים ב־HBase. האינטגרציה בין המערכות מאפשר ל־HBase לשמש במקביל כמקור מידע עבור תוכניות ה־MapReduce וגם כמקום לשמירת התוצאות של השאילתות.

3.3.6 גישה לנתונים ב־MongoDB

MongoDB מחלק את הגישה למסד הנתונים לשני חלקים לוגיים. החלק הלוגי הראשון מגדיר כיצד ניתן להתחבר למסד הנתונים. השפה שהוגדרה ב־MongoDB היא פרוטוקול בשם Mongo Wire Protocol. פרוטוקול זה משתמש בפורמט BSON כדי לתאר את הנתונים העוברים בו, ומגדיר מספר יחסית קטן של הודעות פרוטוקול המאפשרות להעביר פקודות למסד הנתונים ולקבל ממנו תשובות. ניתן לגשת אל מסד הנתונים מכל שפה שניתן באמצעותה לפתוח חיבור בפרוטוקול TCP ושניתן לממש בה את פרוטוקול הגישה של Mongo. נכון לגרסה הנסקרת, קיימים מימושים בשפות רבות כגון C, C#, C++, Java, Ruby, Haskell, PHP, Perl ועוד לפרוטוקול הגישה של MongoDB. החלק הלוגי השני בגישה למסד הנתונים הוא שפת שאילתות המאפשרת לתוכנית להשתמש ביכולות מסד הנתונים. שפת השאילתות שנבחרה ב־MongoDB היא JavaScript. מסד הנתונים מכיל בתוכו מימוש של שפת JavaScript ואוסף של פונקציות הכתובות בשפה זאת המאפשרים שליטה על מסד הנתונים ועל כל המידע שבתוכו.

בנוסף לגישה באמצעות פרוטוקול Mongo Wire Protocol, ניתן להפעיל ע"י פרמטר בקובץ ההגדרות של מסד הנתונים את מסד הנתונים עם תמיכה מובנית לממשק REST מעל פרוטוקול HTTP. באמצעות פרוטוקול זה ניתן לגשת רק לחלק מיכולות מסד הנתונים, אבל מכיוון שהגישה נעשית מעל פרוטוקול http אז לא צריך לפתח או להשתמש בספרייה מיוחדת המממשת את Mongo Wire Protocol.

כחלק מאוסף הפונקציות הקיימות במסד הנתונים במנוע ה־JavaScript הפנימי של מסד הנתונים קיימת ספריית MapReduce המאפשרת הפעלת תוכניות MapReduce על גבי נתונים הנשמרים ב־MongoDB. בנוסף, קיימת אינטגרציה בין MongoDB לבין תשתיות פרויקט Hadoop המאפשרות הרצה

של תוכניות MapReduce ב־Hadoop כך שמקור הנתונים יהיה ב־MongoDB והתוצאות ישמרו אף הן ב־MongoDB.

3.4 אבטחת מאגרי מידע

מודל האבטחה אליו אתייחס הוא מודל CIAA שהוזכר בפרק 1. נבדוק כל אחד מהמאפיינים במודל, ונגדיר את הנדרש ממסד הנתונים כדי שמאפיין זה יתקיים. למטרות ההגדרות בהמשך, משתמש הוא כל גורם היכול לגשת ישירות או בעקיפין אל הנתונים במסד הנתונים. לדוגמה, יתכן ומשתמש הוא מחשב אחר השייך לאותו אשכול, או משתמש ברמת מערכת ההפעלה בעל גישה למערכת הקבצים.

סודיות

מאפיין זה דורש שמשתמש לא מורשה לא יוכל לגשת אל הנתונים. נגדיר משתמש מורשה כמשתמש שזהותו אומתה ע"י מסד הנתונים בדרך כלשהי (נתייחס לבעיות באימות המשתמש בהמשך). על מנת שמאפיין הסודיות ישמר, נדרוש את התנאים הבאים:

1. על מסד הנתונים למנוע גישה למידע מגורמים שלא זוהו בהצלחה, בין אם הגישה מבוצעת באמצעות מנגנוני מסד הנתונים, ובין אם לא
2. על מסד הנתונים לאפשר למנהל מסד הנתונים להגדיר הרשאות עבור משתמשים שמסד הנתונים זיהה בהצלחה
3. מסד הנתונים צריך להגן על נתונים גם מפני מנהל מסד הנתונים

מסדי נתונים רבים ממשים מודל גישה מבחינה (Discretionary Access Control) בו מנהל מסד הנתונים מקצה זכויות גישה עבור משתמשים או תפקידים, ומסד הנתונים מאפשר או מונע מהמשתמש גישה אל הנתונים בהתאם להרשאות שמנהל מסד הנתונים הגדיר עבורו. לדוגמה, בהינתן טבלה בשם Data ומשתמש מערכת בשם User, יכול מנהל מסד הנתונים להרשות למשתמש גישת קריאה בלבד ע"י נתינת הרשאות קריאה ומחיקת הרשאות הכתיבה עבור המשתמש. במסד נתונים רלציוני, הגדרת ההרשאות תראה כך:

```
grant select on Data to User;
revoke update, delete, insert on Data from User;
```

הפקודה הראשונה מאפשרת למשתמש הרשאות קריאה, בעוד הפקודה השנייה שוללת מהמשתמש הרשאות לעדכון הנתונים שהיו לו. בנוסף, מסדי נתונים רלציונים מאפשרים למנהל מסד הנתונים להגדיר מבטים המבוססים

על שאילתות SQL שאינם שולפות את כל הטורים בטבלה, ולמנוע ממשתמשים לא מורשים גישה ישירה אל הטבלה אבל לאפשר גישה אל המבט. בצורה זאת ניתן לנהל הרשאות פרטניות על טבלאות ברמת הטורים.

מסדי נתונים צריכים לאכוף את ההרשאות שהוגדרו על נתונים מרגע שפעולת עדכון ההרשאות הסתיימה, וללא תלות במחשב שעליו ההרשאות האלה הוגדרו באשכול של מסד הנתונים.

שלמות

מאפיין זה דורש שרק משתמשים מורשים יכולים לעדכן מידע. על מנת שמאפיין זה ישמר, נדרוש את התנאים הבאים:

1. על מסד הנתונים לאפשר הגדרת הרשאות למשתמשים
2. על מסד הנתונים למנוע פעולות עדכון במידה ואין למשתמש הרשאות לביצוע פעולה
3. במידה ופעולה לא חוקית בוצעה, על מסד הנתונים לזהות זאת ולהתריע על כך

דרישה זאת במידה רבה משלימה את דרישת הסודיות. בעוד שדרישת הסודיות התייחסה בעיקר ליכולת שליפת הנתונים של משתמש, דרישה זאת מתייחסת ליכולת של משתמש לעדכן נתונים, ויכולת מנהל מסד הנתונים לשלוט בהרשאות העדכון הניתנות לכל משתמש במסד הנתונים.

זמינות

מאפיין זה דורש שמשתמש יוכל לבצע כל פעולה שיש לו הרשאות לבצעה בכל רגע שפעולה זאת נדרשת. נגדיר את התנאים הבאים:

1. על מסד הנתונים להיות זמין למשתמשים חוקיים בכל שלב
2. על מסד הנתונים לאפשר למנהל מסד הנתונים להגדיר עבור כל משתמש את אסופת ההרשאות עבור כל פריט מידע במסד הנתונים

זמינות של מסד נתונים מבוזר יכולה להימדד גם ביכולתו להיות זמין למשתמשים לגיטימיים כאשר תוקף מפעיל נגד מסד הנתונים התקפת Denial Of Service.

אימות

מאפיין זה דורש שניתן יהיה לאמת שכל משתמש הוא אכן מי שהוא אמור להיות. נגדיר את התנאים הבאים:

1. לפני קבלת כל שירות ממסד הנתונים, על מסד הנתונים לזהות את מבקש השירות בכל דרך המתאימה למסד הנתונים

2. נדרוש שהגדרת משתמש תהיה עקבית ושינוים בהגדרות המשתמשים יילקחו בחשבון באופן זהה ללא הבדל בנקודת הגישה

צורת אימות המשתמש צריכה להיות אמינה וקשה לזיוף או לביצוע ע"י משתמש מתחזה או תוקף. שיטת ההזדהות הבסיסית ביותר היא הצגת שם משתמש וסיסמא למסד הנתונים. חיסרון מהותי של השיטה הבסיסית הזאת הוא ששימוש בסיסמה מאפשר למסד הנתונים לאמת את זהות המשתמש המתחבר, אבל איננו מבטיח למשתמש שהוא אכן התחבר אל מסד הנתונים ולא אל מחשב של תוקף. ישנם פרוטוקולי אימות קריפטוגרפיים המאפשרים לשני הצדדים - מסד הנתונים והמשתמש המתחבר אליו - לאמת את זהותם אחד לשני. דוגמאות לפרוטוקולים כאלה הם פרוטוקול SSL [26] או פרוטוקול Kerberos [27]. SSL מאפשר אימות ע"י שימוש בתעודות (X509 Certificates) המכילות את הזהות של הצד המציג אותם וחתומות ע"י גורם שלישי (Certificate Authority) שעליו שני הצדדים סומכים. אין צורך שבשלב האימות הגורם השלישי יהיה זמין. אימות הזהות נעשה ע"י ווידוא החתימה דיגיטלית שעל התעודה שכל אחד מהצדדים מציג. פרוטוקול Kerberos דורש שכל אחד מהצדדים יאמת את זהותו לשרת שלישי (Key Distribution Center או KDC), וכל צד שרוצה לקבל שירות יכול לקבל תעודה (Ticket) משרת ה-KDC ופעולת האימות נעשית ע"י הצגת התעודה הזאת לשרת ממנו נדרש השירות. התעודה הניתנת ע"י פרוטוקול Kerberos למטרת אימות היא בעלת אורך חיים קצר יחסים, הנמדד בשעות, לעומת תעודה המשמשת את פרוטוקול SSL שאורך חייה נמדד בדר"כ בשנים, ולכן פרוטוקול Kerberos איננו רגיש כ"כ למצב בו השרת השלישי חדל להיות גורם אמין. לדוגמה, בשנת 2011 התגלה ש-Certificate Authority בשם DigiNotar נפרץ ע"י תוקפים שהפיקו תעודות תקינות שאפשרו לשרתים תוקפים לבצע אימות מוצלחת שהיה צריך להיכשל [28].

לא נדרוש שמסד הנתונים עצמו יממש את מנגנון האימות. בהחלט יתכן שמסד הנתונים יפנה אל מערכת ההפעלה עליה מסד הנתונים מופעל, או אל שרת או שירות אחר הזמין למסד הנתונים, כדי לבצע את פעולת האימות. לדוגמה, שרת Kerberos המותקן ברשת המקומית מבצע את האימות, ומסד הנתונים מסתמך על התעודה ששרת ה-Kerberos מספק, מבלי שמסד הנתונים יידרש לממש את שירות ה-KDC עצמו.

3.4.1 נקודות הבדיקה

עבור כל אחד ממסדי הנתונים הנבדקים אבדוק את הנקודות הבאות שנסקרו למעלה:

סודיות שיטת ההגנה שמסד הנתונים מאפשר על מידע במנוחה (בקבצים המקומיים) ושיטת ההגנה על מידע בתנועה (אל האפליקציה או אל מחשב אחר באשכול), ומנגנון ההרשאות שמסד הנתונים מספק כדי לאפשר למנהל מסד הנתונים להגדיר את סודיות הנתונים.

שלמות שיטת הווידוא של מסד הנתונים שמידע במנוחה לא שונה מחוץ למסד הנתונים במנוחה (לדוגמה בקבצי הנתונים) ובתנועה (לדוגמה ע"י שינוי של הודעות ברשת) ובדיקת יכולת מסד הנתונים להגדיר מהו עדכון חוקי שישאיר את המידע שלם.

זמינות הצגת המנגנונים במסד הנתונים המאפשרים למסד הנתונים להתאושש מכישלונות של האשכול, ובדיקת יכולות מסד הנתונים להימנע מכישלון של רכיבים באשכול.

אימות בדיקת מנגנון האימות הקיים במסד הנתונים, והצורה בה מסד הנתונים מוודא שהתקשורת אליו מבוצעת ע"י גורם מורשה - תוכנה חוקית או חבר אחר באשכול. בדיקת יכולת מסד הנתונים להימנע מהקצאת משאבים רבים לטיפול במתחזים.

4 שיטות העבודה

כדי לבצע את סקירת האבטחה, התקנתי את כל סוגי מסדי הנתונים השונים בסביבה וירטואלית מסוג Proxmox [29] ובנוסף התקנתי את כל השירותים הנוספים הנדרשים כדי לאבטח את מסדי הנתונים הללו. שירותים אלה כללו שרתי שמות DNS כתשתית בסיסית, שרתי LDAP ו־Kerberos למטרות הגדרת וזיהוי משתמשים. באיור 8 מתוארת סביבת המעבדה הווירטואלית בה אבצע את הבדיקה.

עבור סביבת HBase over Hadoop, הגדרתי אשכול Hadoop שכלל שלושה חברים מסוג DataNode, ושני שרתים ששמשו NameNode ראשי ומשני. בסביבה זאת התקנתי גם את שרתי מסד הנתונים עצמו, כאשר האחסון של הנתונים מנוהל בתוך Hadoop. מנגנון האבטחה של מסד הנתונים הופעל, ולצורך זה התקנתי שרת ה־Kerberos שמולו בוצעו פעולות ההזדהות.

עבור סביבת Cassandra התקנתי צביר בגודל שלוש מכונות וירטואליות. מסד הנתונים אובטח עפי המוסבר במדריך למשתמש של התוכנה.

עבור סביבת MongoDB התקנתי שני Replica Sets, כאשר כל Replica הוגדר כ־shard נפרד באשכול גדול יותר. בנוסף, התקנתי שרת ניתוב ושרת קונפיגורציה לאשכול.

לאחר שהתקנתי את כל הסביבות, בדקתי את כל התכונות שהוגדרו בפרק 3.4 והשוואתי בכל מקרה למסד הנתונים רלציוני בשם PostgreSQL המפותח כפרויקט קוד פתוח בעל תכונות אבטחה רבות ומייצגות ליכולות של מסדי נתונים רלציונים הזמינים לפרויקטים המפותחים היום. בכל המקרים, מסדי הנתונים הותקנו בסביבת Debian GNU/Linux עדכנית, והקונפיגורציה הותאמה כך שכל שינויי ההקשחה המומלצים ע"י הפרויקט בוצעו כדי לשפר את האבטחה.

עבור Cassandra אני מימשתי תכונות אבטחה עבור אימות זהות המשתמשים

איור 8: סביבת המעבדה

והרשאות המקיימות את הנדרש בפרק 3.4 שאינם מסופקות עם מסד הנתונים, וזה מתואר בפרק 6.

5 הצגת התוצאות וניתוחם

5.1 Cassandra

כברירת מחדל, מסד הנתונים איננו מאובטח כלל. בתצורה הקיימת מיד אחרי ההתקנה, מסד הנתונים מכיל מימוש של מנהל משתמשים ומנהל הרשאות המאפשרים לכל לקוח המתחבר אל מסד הנתונים לבצע כל פעולה על כל אחד מהממשקים שמסד הנתונים מספק, הן לאפליקציות והן למחשבים אחרים בצביר. בגרסאות החדשות יותר של מסד הנתונים, אין מימוש אחר של מנהלי הרשאות ומשתמשים ולא ניתן להגדיר משתמשים והרשאות. מסד הנתונים חושף את האפשרות להשתמש במנהלי הרשאות ומשתמשים אחרים, בהתאם לצורך, ע"י הוספת מימוש לשני ממשקים (interfaces) בשפת Java המממשים את הנדרש כדי לנהל הרשאות ומשתמשים. ישנם דוגמאות למימוש פשוט של מנהלי הרשאות ומשתמשים הניתנים להורדה מאתר הפרויקט. הדוגמאות המסופקות מהאתר היו זמינות בגרסאות ישנות יותר כחלק ממסד הנתונים, אבל בגלל שהם אינם נחשבים ע"י מפתחי הפרויקט כמספיק טובים, הם הוצאו ממסד הנתונים והועברו לאתר הפרויקט כדוגמאות. הבעייתיות בדוגמאות המסופקות נובעת מכך שהדוגמאות מעבירות את הסמאות של המשתמשים בצורה גלויה על גבי התקשורת אל מסד הנתונים, ובכך שצריך לסנכרן ידנית את קבצי ההרשאות בין כל החברים בצביר של מסד הנתונים. מודל ההרשאות הנתמך במסד הנתונים מאפשר להגדיר כל צירוף של הרשאות קריאה, עדכון, תיאור, יצירה, מחיקה ושינוי עבור כל Keyspace וכל Column Family לפי הפירוט המופיע בטבלה 1.

בתצורתו הבסיסית בה אין שימוש בהרשאות במסד הנתונים יכול כל משתמש לקרוא ולכתוב בכל מרחב מפתחות וכל משפחת טורים. בפרט, מרחב המפתחות system המשמש את המערכת כקטלוג ניתן לעדכון. אם בוצע עדכון שגוי של משפחות טורים במרחב system, מסד הנתונים איננו יודע להתאושש והמידע במסד הנתונים יכול להאבד.

Cassandra מזהה כל משאב במערכת ע"י תווית המורכבת מארבעה חלקים:

$$/ \ cassandra \ / \ keyspaces \ / \ \$keyspaceName \ / \ \$CFName$$

החלק הראשון מגדיר שמחרוזת זאת מייצגת משאב במסד הנתונים. החלק השני מייצג את כל מרחבי המפתחות במערכת, והרשאות הניתנות על חלק זה מאפשרות למשתמש לקרוא את רשימת מרחבי המפתחות, להוסיף ולמחוק מרחבי מפתחות. החלק השלישי מייצג מרחב מפתחות מסוים, והרשאות

שם הרשאה	רלוונטי עבור	פעולות הדורשות את ההרשאה
CREATE	מרחבי מפתחות ומשפחות טורים	יצירת מרחב מפתחות ויצירת משפחת טורים
DROP	מרחבי מפתחות ומשפחות טורים	מחיקת מרחב מפתחות ומחיקת משפחת טורים
ALTER	מרחבי מפתחות ומשפחות טורים	עדכון תכונות של מרחבי מפתחות ומשפחות טורים, יצירת ומחיקת אינדקסים
DESCRIBE	מרחבי מפתחות	הצגת משפחות טורים המוגדרות במרחב מפתחות
UPDATE	משפחות טורים	עדכון והוספה של מידע במשפחות טורים
DELETE	משפחות טורים	מחיקת מידע ממשפחות טורים
SELECT	משפחות טורים	שליפת מידע ממשפחות טורים
FULL_ACCESS	הכל	מאפשר גישה מלאה לכל משאבי מסד הנתונים
NO_ACCESS	הכל	הרשאה הופכית של FULL_ACCESS

טבלה 1: הרשאות ב־Cassandra

הניתנות על חלק זה מאפשרות למשתמש לקרוא את רשימת משפחות הטורים במרחב מפתחות, ולהוסיף או למחוק משפחות טורים במרחב מפתחות. החלק הרביעי מייצג את ההרשאות על משפחת טורים מסוימת, והרשאות ברמה זאת מאפשרות קריאה וכתיבה או מחיקה של מידע במשפחת הטורים. לדוגמה, נתינת הרשאת CREATE על המשאב /cassandra/keyspaces/test תאפשר למשתמש להוסיף משפחות טורים במרחב המפתחות הנקרא test.

חשוב להדגיש שלמרות שמסד הנתונים מגדיר את מנגנון ההרשאות שתואר, מסד הנתונים איננו מספק מימוש למודל ההרשאות, וכדי שמודל ההרשאות יעבוד יש צורך לכתוב קוד המממש את עצם חיבור ההרשאות למשתמש מערכת שזוהה.

מסד הנתונים מאפשר בקונפיגורציה פשוטה להגדיר הצפנה של כל התעבורה בין מחשבים בצביר. מנהל המערכת צריך לספק לכל חבר בצביר תעודת SSL (ומפתח פרטי מתאים) שתשמש לזיהוי של החבר, ועוד תעודת SSL שבאמצעותה ניתן לאשר את זהות מחשבים אחרים בצביר המנסים להתחבר את המחשב. מנגנון זה בעצם מאפשר סוג מנוון של אימות משתמשים מורשים עבור ממשק ה־Storage API הפנימי של Cassandra. המנגנון מניח שמנהל המערכת מנהל בנוסף לצביר גם X509 certificate authority ובאמצעות מנגנון זה, כל אפליקציה חוקית מקבלת תעודה החתומה ע"י ה־CA. המימוש הקיים בגרסה הנבדקת של מסד הנתונים איננו תומך בשימוש ברשימות תעודות מחוקות (certificate revocation list) וכן איננו מאפשר דרישת תעודה מזהה מהצד המתחבר, ולכן אם תעודה שסופקה ללקוח כלשהו נגנבה, מנהל

המערכת חייב להחליף את כל התעודות של כל האפליקציות החוקיות.

קבצי הנתונים על הדיסק נשמרים כשאינם מוצפנים, ותוקף המשיג גישה אל הקבצים האלה מסוגל לעקוף את כל מנגנוני ההרשאות של מסד הנתונים ע"י בדיקת תוכן הקבצים ישירות ממערכת ההפעלה. מכיוון שהקבצים אינם מוצפנים, אם תוקף מצליח לשנות את קבצי הנתונים בצורה חוקית, ולגרום למסד הנתונים לטעון מחדש את הקבצים, אז תוקף יכול גם לשנות את המידע במסד הנתונים.

התקשורת בין מסד הנתונים לאפליקציות המשתמשות בממשק ה־Thrift איננו מוצפן ולא קיימת במסד הנתונים שיטה לגרום לו להיות מוצפן, ולכן כל תוקף המסוגל לראות באופן פסיבי את התעבורה יכול לגנוב מידע ממסד הנתונים. בהוראות ההתקנה ממליצים למתקינים להשתמש בטכנולוגיות הצפנה חיצוניים למסד הנתונים כדי להצפין את התקשורת בין מסד הנתונים לאפליקציה, לדוגמה ע"י שימוש ב־IPSec או SSL tunnelling.

ממשק ה־CQL של Cassandra חשוף לבעיות מסוג הזרקת פקודות command injection בצורה זהה למסדי נתונים רלציונים רגילים, ולכן על אפליקציה המשתמשת בממשק זה לדאוג לווידוא הקלט, באופן דומה לנדרש מאפליקציה המשתמשת במסד נתונים רלציוני רגיל ושפת SQL. ממשק ה־CQL תומך בהפרדת הוראות ומידע ע"י שימוש ב־prepared statements באופן דומה לרוב מסדי הנתונים הרלציונים ע"י שימוש בסט הודעות Thrift להפעלת CQL המתאימים (prepare_cql_query ו־execute_prepared_cql_query במקום ב־execute_cql_query). באופן זה ניתן למנוע מקלט לא תקין להתפענח כחלק מפקודת ה־CQL. השימוש במנגנון הזה (בדומה למצב במסדי נתונים רלציונים) הוא באחריות האפליקציה, ולא ניתן לחייב אפליקציה להשתמש בו.

לסיכום:

סודיות הגנה על קבצי הנתונים של מסד הנתונים מסתמכת על מערכת ההפעלה שעליה מותקן מסד הנתונים. יש אפשרות להצפין את כל התעבורה בין שרתים שונים באשכול, אבל לא ניתן להצפין את התעבורה בין האפליקציה למסד הנתונים עצמו. מסד הנתונים תומך במנגנון הרשאות, אבל איננו מספק מימוש מובנה המאפשר להגדיר הרשאות על אובייקטים במסד הנתונים.

שלמות מסד הנתונים איננו תומך במנגנון כלשהו לבדיקת שלמות הנתונים.

זמינות רמת הזמינות שמסד הנתונים מספק תלויה בכמות השרתים שהותקנו, וברמת הזמינות שהאפליקציה הגדירה. האפליקציה קובעת בהתאם לצרכיה את מספר ההעתקים שמסד הנתונים ישמור עבור כל פריט מידע. כל עוד לפחות שרת אחד המכיל את המידע הנדרש זמין, המידע יהיה זמין. מסד הנתונים תומך בתצורה שיכולה להיות מבוזרת במספר אתרים גאוגרפים שונים כדי להגדיל את זמינות הנתונים.

אימות מסד הנתונים מכיר בכך שיש משמעות לזהות הישות המתחברת אליו,

אבל איננו מספק מנגנון עובד לביצוע אימות. מנהל מסד הנתונים יכול להוסיף קוד ל־Cassandra המבצע את פעולת האימות בהתאם לנדרש בהתקנה מסויימת.

5.2 HBase over Hadoop

מכיוון ש HBase בתצורתו המבוזרת משתמש ב־Hadoop כבדיסק מבוזר, לפני שניתן לאבטח את HBase, צריך לאבטח את Hadoop. כברירת מחדל, מערכת הקבצים HDFS של Hadoopמאפשרת הגדרת הרשאות בצורה דומה להרשאות הקיימות במערכות קבצים של Unix, כאשר המשתמש האפקטיבי הוא המשתמש המריץ את התהליך של Hadoop במערכת ההפעלה. ניתן לשנות את ההגדרה של Hadoop כך שמערכת הקבצים תעבוד מול שרת Kerberos. בתצורה זאת, המשתמש האפקטיבי עבור כל פעולה המתבצעת מול HDFS הוא המשתמש שאומת ע"י Kerberos. בנוסף, כל תהליך של Hadoop חייב להזדהות בתצורה זאת לכל תהליך אחר המורץ במחשב מרוחק ע"י הצגת token חוקי שאושר ע"י שרת ה־KDC. ישנם מספר בעיות במודל האבטחה של HDFS:

1. למרות שאימות המשתמש מבוצע באמצעות Kerberos, שיוך המשתמשים לקבוצות מבוצע ע"י הפעלת פקודת groups של מערכת ההפעלה בשרת ה־namenode של הצביר, כאשר שם המשתמש הוא הפרמטר לפקודה. התוצאה היא שאם לתוקף יש גישה לשרת זה, התוקף יכול לשנות את רשימת הקבוצות של המשתמשים ב־HDFS ע"י מניפולציה של פקודת groups. ישנה קבוצה בשם supergroup המאפשרת למשתמשים המשויכים אליה גישה מלאה לכל הקבצים השמורים ב־HDFS.

2. אתר הניהול של שרתי הצביר משתמשים גם הם ב־Kerberos באמצעות פרוטוקול SPNEGO כדי לבצע אימות למשתמש. לאחר ביצוע האימות, מוקצה למשתמש cookie עבור ה־session שלו, ולא מבוצע אימות נוסף באמצעות Kerberos. מכיוון שהתקשורת לאתר הניהול של HDFS איננה מוצפנת, תוקף המסוגל להאזין לתקשורת יכול לגנוב את ה־cookie ולהשתלט על ה־session. מנקודה זאת, ממשק הניהול של HDFS יאפשר גישה מלאה לכל הקבצים לפי הרשאות המשתמש שהרשאותיו נגנבו.

3. תקשורת ה־RPC של HDFS איננה מוצפנת ולכן כל תוקף המסוגל להאזין לתקשורת יכול לראות את כל המידע המועבר לתוך ומתוך מערכת הקבצים המבוזרת.

כאשר HDFS הוגדר עם אימות מבוסס Kerberos, התקנת HBase צריכה גם היא לאמת את זהותה לפני שמסד הנתונים HBase יכול להתחבר למערכת הקבצים. הפעלת יכולת זאת ב־HBase תגרום למסד הנתונים להשתמש ב־ticket שמשתמש למטרות אימות הרשאות עבור פעולות המבוצעות מול מסד

הנתונים. מסד הנתונים עצמו משתמש גם הוא ב־Kerberos כדי לאמת את זהותו לשאר תהליכי HBase. הגישה למערכת הקבצים נעשית באמצעות משתמש מערכת, ולא באמצעות ה־ticket של המשתמש המתחבר ל־HBase. בנוסף לגישה אל קבצי מסד הנתונים הנשמרים על HDFS, יש צורך לאמת את הגישה אל מנהל התיאום (ZooKeeper). אימות זה מבוצע ע"י שימוש בפרוטוקול SASL [30] (פרוטוקול המפריד בין מנגנון אימות הזהות מהפרוטוקול של השירות הדורש את פעולת האימות) עם Kerberos כמנגנון ההזדהות.

HBase מאפשר הגדרת הרשאות ברמת טבלאות וברמת משפחות טורים, כאשר ברמת משפחות טורים ההרשאות האפשריות הן קריאה וכתיבה בלבד. ההרשאות הניתנות להגדרה הן:

- קריאה ־ משתמש עם הרשאה זאת יכול לקרוא את תוכן הטבלה.
- כתיבה ־ משתמש עם הרשאה זאת יכול לעדכן את תוכן הטבלה.
- יצירה ־ משתמש עם הרשאה זאת יכול לשנות את הגדרת הטבלה ולהוסיף או להוריד משפחות טורים מהטבלה. בנוסף הרשאה זאת מאפשרת למחוק את הטבלה.
- ניהול ־ הרשאה זאת זהה להרשאת יצירה, ומאפשרת בנוסף פעולות שפעול וביטול (enable, disable) טבלאות, כדי להפוך את הטבלאות לזמינות או לא זמינות.

אם משתמשים בשרתי גישה אל HBase כדי לאפשר גישה באמצעות Thrift, AVRO או REST אז כל יתרונות האימות ומנגנון ההרשאות מבוטלות. בתצורה זאת, האימות אל HBase מבוצע ע"י משתמש שהוגדר בשרתי הגישה, ולא באמצעות משתמש הקצה מהאפליקציה המתחברת אל מסד הנתונים. בתצורה זאת על האפליקציה לבצע את כל פעולות האימות וניהול ההרשאות. אם משתמשים בפרוטוקול ה־RPC של HBase אז ניתן להשתמש בכל שירותי האימות והאישור של מסד הנתונים, ובנוסף (ובניגוד לפרוטוקול ה־RPC של Hadoop) התעבורה בין האפליקציה אל מסד הנתונים מבוצעת עם הצפנה המונע מתוקף היכול לראות את התעבורה לראות את המידע העובר על גבי התווך.

כדי לאבטח בצורה טובה את מסד הנתונים, מומלץ לוודא שאין ללקוחות של מסד הנתונים גישה ישירה אל שרתי ה־HDFS אלא רק אל שרתי HBase המשתתפים בצביר, ושמסד הנתונים אכן הוגדר בתצורה המשתמשת באבטחה באמצעות Kerberos, הצפנת התעבורה וניהול הרשאות. בתצורה זאת יש ל־HBase יכולות אבטחה טובות המספקות סודיות ואימות טובים.

הארכיטקטורה של HBase מאפשרת זמינות גבוהה, כאשר לכל רכיב בארכיטקטורה יש מנגנונים המאפשרים למסד הנתונים להתאושש מנפילות. מנהל התאום ZooKeeer מורכב בעצמו מצביר המסוגל לעבוד כל עוד יש לפחות שרת אחד זמין. ניתן בארכיטקטורת HBase להעלות מספר גדול של שרתי

Master כאשר אם הצביר מזהה שה־Master הנוכחי איננו זמין אז אחד אחר יהפוך ל־Master באופן אוטומטי. ניתן להעלות מספר גדול של Region Servers כאשר כל עוד ה־Master זמין אז עליה או קריסה של Region Server אחד לא ישפיע על זמינות המידע, מכיוון שהמנהל יכול להעביר אחריות עבור כל טבלה לכל אחד מה־Region Servers בהתאם לצורך. זמינות המידע מובטח מהשימוש בתשתית HDFS, המספק ארכיטקטורת Shared Disk זמינה ל־HBase.
לסיכום:

סודיות HBase מתבסס על מנגנון ההרשאות על קבצים הבסיסי הקיים ב־Hadoop המזכיר את מנגנון ההרשאות הבסיסי של מערכות קבצים במערכות Unix. התעבורה בין HBase לאפליקציות המשתמשות במנגנון ה־RPC של HBase מוצפן. מסד הנתונים מספק מנגנון הרשאות בסיסי בעל ארבע סוגי הרשאות.

שלמות המידע השמור ב־HBase נשמר ביותר בשרת אחד, ומנגנון התשתית של Hadoop מבצע checksum בסיס על כל בלוק של נתונים הנשמר בתוכו, כדי לזהות בלוקים שזובלו.

זמינות לכל רכיב במסד הנתונים יש גיבוי, ואין אף רכיב המהוה נקודת כשל במסד הנתונים.

אימות נעשה שימוש בשירותי Kerberos למטרת אימות זהות המתחברים אל מסד הנתונים.

5.3 MongoDB

כברירת מחדל, מסד הנתונים איננו מאובטח כלל. בתצורה הקיימת מיד אחרי התקנה, אין אפשרות להגדיר משתמשים או הרשאות, ולא נעשה אף ניסיון להגן על המידע הנשמר במסד הנתונים. אבטחת מסד הנתונים נעשית ע"י הוספת פרמטר auth או פרמטר keyFile לקובץ הקונפיגורציה של מסד הנתונים. לאחר הוספת אחד הפרמטרים האלה, מאפשר מסד הנתונים למנהל מסד הנתונים להגדיר משתמשים במסד הנתונים. הפרמטר הראשון מחייב אפליקציות המתחברות אל מסד הנתונים לבצע פעולת זיהוי באמצעות משתמש שהוגדר במסד הנתונים, והפרמטר השני מרחיב את הדרישה לזיהוי גם למחשבים השייכים לאשכול, והוא נדרש אם מסד הנתונים מורץ בתצורת אשכול. הפרמטר keyFile גורר באופן אוטומטי את הפרמטר הראשון. הפעלת מסד הנתונים באמצעות keyFile מחייב הוספת קובץ על כל אחד מהמחשבים השותפים בצביר ובו סיסמא עבור הצביר. הקובץ איננו נשמר מוצפן, והתיעוד של מסד הנתונים ממליץ להגן על הקובץ באמצעות מנגנוני הרשאות של מערכת ההפעלה המארחת. תוכן הקובץ הוא טקסט אקראי המייצג מחרוזת חוקית בייצוג BASE64 באורך הקטן מ־1024 תווים. מסד הנתונים משתמש בתוכן הקובץ כסיסמא למשתמש מערכת בשם __system המשמש לפעולות

הזדהות של תהליכים הרצים במחשבים נפרדים. מבדיקת התעבורה עולה שמנגנון ההזדהות של מחשב אחד בצביר לחברו מתבצע בדיוק באותה צורה בה מזדהים משתמשים רגילים למסד הנתונים. שיטת ההזדהות מפורטת בהמשך. אם תוקף משיג את קובץ הסיסמא, אז התוקף יכול להוסיף שרת לאשכול, לדוגמה שרת mongos. כל אפליקציה חוקית המתחברת לאשכול דרך שרת זה תהיה חשופה להתקפות ולהפרעות.

לא קיימת במסד הנתונים אף תכונת אבטחת נתונים אחרת המובנת לתוך מסד הנתונים. בתיעוד מסד הנתונים ממליצים המפתחים לדאוג לאבטח את הרשת ואת מערכת ההפעלה המארחת ע"י שימוש בחומות אש ברשת, ומנגנונים של מערכת ההפעלה. בפרט, מסד הנתונים כפי שניתן להורדה מאתר הפרויקט איננו תומך בהצפנת התעבורה ברשת (ניתן לקנות גרסה התומכת בהצפנת התעבורה באמצעות SSL או להדר בקוד המקור את הגרסה הזאת). המשמעות היא שבשימוש בגרסת מסד הנתונים הסטנדרטית, כל תוקף המסוגל להאזין בצורה פסיבית לתעבורה ברמת הרשת מסוגל לראות בצורה פשוטה את כל פעולות האפליקציות המשתמשות במסד הנתונים, וכן את כל פעולות הסנכרון המתבצעות בין חברי האשכול של מסד הנתונים. מסד נתונים זה איננו תומך בדרישת הסודיות, אלא מניח שהבעיה מטופלת ברמה אחרת.

המידע הנשמר במסד הנתונים איננו נשמר מוצפן, ולא קיימת אפשרות להכריח את מסד הנתונים להצפין את המידע. אם תוקף הצליח להגיע אל קבצי הנתונים של מסד הנתונים, אז התוקף יכול לקרוא אותם (הקבצים נשמרים בפורמט הדומה ל־BSON) ולראות את כל המידע הנשמר באותו השרת. מסד הנתונים איננו מכיל אף בדיקה על תוכן קבצי הנתונים, מעבר להיותם במבנה תקין. תוקף המשנה את קבצי הנתונים שלא דרך מנגנוני מסד הנתונים יכול לגרום למסד הנתונים להציג מידע שגוי. כמובן שאם תוקף משנה את קבצי הנתונים ישירות על הדיסק, עליו לעשות זאת בכל המחשבים ב־replica set, כדי לוודא שאפליקציה תקבל מידע שגוי.

מסד הנתונים מצטיין ביכולות הזמינות שלו. בכל replica set המוגדר במסד הנתונים, מסד הנתונים מסוגל להמשיך לעבוד כל עוד יש לפחות מחשב אחד זמין. אם אפליקציה יודעת שהיא איננה צריכה לעדכן את מסד הנתונים, האפליקציה יכולה להתחבר לכל אחד מהמחשבים המשניים ב־replica set כדי לקרוא מידע, ובכך להגדיל את כמות האפליקציות המסוגלות לקבל מידע ממסד הנתונים. מסד הנתונים מאפשר להגדיר צביר של עד שלושה שרתי פיזור נתונים (config servers) בצביר מסוג shard, ומסד הנתונים ימשיך לעבוד כל עוד לפחות שרת אחד כזה זמין.

הגדרת משתמשים ברמת האפליקציה במסד הנתונים נעשה ע"י התחברות לסכמה בשם admin והפעלת שגרת Javascript במנוע הפנימי של מסד הנתונים, המאפשרת להגדיר שם משתמש, סיסמא, והגדרת הרשאות קריאה־כתיבה או קריאה־בלבד עבור המשתמש שהוגדר. הגדרת המשתמש נשמרת במסד הנתונים באוסף מסמכים הנקרא system.users. שם המשתמש נשמר לא

מוצפן, והסיסמא שהוגדרה נשמרת לאחר שעברה גיבוב באלגוריתם MD5 עם שם המשתמש משורשר לתחילתה כדי להמליח את הגיבוב בצורה הבאה:

$$MD5(username||" : mongo : "||password)$$

להלן דוגמה להוספת משתמש והצגת הרשומה והסיסמא המגובבת של המשתמש.

```
mongos> use admin
switched to db admin
mongos> db.auth("root", "123456")
1
mongos> db.addUser("lior", "testpassword", true);
mongos> db.system.users.find({ "user" : "lior"})
{ "_id" : ObjectId("5061ddf0b40b36037a6b1068"),
  "user" : "lior",
  "readOnly" : true,
  "pwd" : "4d43387826be6231792b128b08df2b44" }
mongos> hex_md5("lior:mongo:testpassword")
4d43387826be6231792b128b08df2b44
```

בבדיקת התעבורה בזמן ביצוע הפקודות הללו, רואים סדרות של זוגות הודעות רשת נפרדות עבור כל פקודה שבוצעה. פעולת הזדהות (db.auth) נראית כך בהודעות רשת:

```
192.168.2.50:49881  —>> 192.168.2.50:27017 admin.$cmd
        query: { getnonce: 1 }
192.168.2.51:27017  <<—  192.168.2.51:49881
        reply n:1 cursorId: 0
                { nonce: "fd6d168bf09a30ce", ok: 1.0 }
192.168.2.50:49881  —>> 192.168.2.50:27017 admin.$cmd
        query: {authenticate: 1,
                nonce: "fd6d168bf09a30ce",
                user: "root",
                key: "3727eda58fc450a3975f49925c7fe6bd" }
192.168.2.51:27017  <<—  192.168.2.51:49881
        reply n:1 cursorId: 0
```

```
{ dbname: "admin", user: "root",
  readOnly: false, ok: 1.0 }
```

זוג ההודעות הראשונות מחליפות מספר אקראי nonce בין האפליקציה המזדהה ומסד הנתונים. זוג ההודעות השני מבצע את פעולת ההזדהות עצמה. פעולת ההזדהות מבוצעת ע"י חישוב ערך הנקרא key בצד הלקוח ושליחתו למסד הנתונים. מסד הנתונים מבצע חישוב זהה, וההזדהות מצליחה אם שני הצדדים מגיעים לאותה התוצאה. ערך המפתח מחושב כך:

$$key = MD5\left(nonce||username||MD5\left(username||" : mongo : "||password\right)\right)$$

בשיטה זאת, הסיסמא לא עוברת בצורה גלויה בהודעות ברשת. מכיוון שיש שימוש ב־nonce, אין סכנה מהתקפת שידור־חוזר, מכיוון שערך מפתח הטוב לפעולת הזדהות אחת לא יתאים לפעולת הזדהות שנייה. כדי לפרוץ את מנגנון ההזדהות של מסד הנתונים בצורה ישירה על תוקף לבצע שתי התקפות pre-image collision על אלגוריתם MD5, ולכן פעולת האימות איננה פגיעה בצורה ישירה. לרוע המזל, אלגוריתם ההזדהות הזה כן ניתן לפריצה באמצעות התקפת מילון. תוקף המשיג את ההודעה השלישית, המכילה את ערך המפתח, שם המשתמש וה־nonce שבו נעשה שימוש יכול לנסות לייצר גיבובים בהם התוקף משנה את הסיסמא ובודק אם לאחר חישוב המפתח התקבל הערך ששודר למסד הנתונים. ההתקפה תבוצע כך:

```
for each password in the dictionary
        guess := MD5(username || ':mongo:' || password)
    test := MD5(nonce||username||guess)
    if test == key then print current password
end for
```

התקפת מילון כזאת איננה תוקפת את אלגוריתם הגיבוב אלא את הסיסמא שנבחרה ע"י משתמשי מסד הנתונים, ויש חשד סביר להניח שהיא תצליח. יכולות אימות המשתמשים של מסד נתונים זה איננה חזקה במיוחד, ומסתמכת בסופו של דבר על חוזק הסיסמא שבה בחר המשתמש.

אם מסד הנתונים הופעל עם תמיכה בארכיטקטורת REST דרך הדפדפן, המצב הרבה יותר גרוע. במקרה כזה, ה־nonce המועבר לשימוש בהזדהות של פרוטוקול html הוא תמיד הערך "abc". מסיבה זאת, ממשק ה־REST של מסד הנתונים ניתן לתקיפה גם ע"י התקפת שידור־חוזר של התעבורה. מכיוון שמסד הנתונים איננו תומך בהצפנת התעבורה בממשק זה, כל תוקף

הרואה תעבורה מעל ממשק REST יכול מיד להתחבר אל מסד הנתונים ברמת ההרשאות של המשתמש שאת התעבורה שלו התוקף ראה.

MongoDB החל דרכו כמסד נתונים לאתרי אינטרנט, ולכן השימוש בשפת Javascript כשפת גישה אל מסד הנתונים נראה מתאים במיוחד - רוב עבודת הפיתוח של אתרים באינטרנט נעשה ע"י מפתחים המכירים את שפת Javascript. השימוש בשפה הכנה למסד הנתונים יכולות חזקות מאוד בדמות שפת scripting מובנת, ומאפשר העברת חלק מהעיבוד מהאפליקציה אל מסד הנתונים. יש בכל גם בעיית אבטחה, מכיוון שאם תוקף מצליח לגרום למסד הנתונים לבצע קטע קוד במסד הנתונים, התוקף יכול לגרום למספר בעיות אבטחה. לדוגמה, הפעלת פונקציית sleep של Javascript יכולה לגרום למסד הנתונים להפסיק לטפל במשתמשים אחרים. לכן MongoDB יכול לסבול מבעיית התקפות Javascript injection, בדומה לבעיית SQL injection המוכרת מעולם מסדי הנתונים הרלציונים. הפתרון, כמו בעולם מסדי הנתונים הרלציונים, הוא שאפליקציות הניגשות למסד הנתונים יוודאו את הקלט שמשתמשיהם מזינים, ולעולם לא יעבירו את הקלט הזה למסד הנתונים בצורה המסכנת את מסד הנתונים. הבעיה ב־MongoDB חמורה מהבעיה במסדי נתונים רלציונים, מכיוון שאין תשתית סטנדרטית ב־MongoDB לטיפול בקלט, המקביל לתשתית ה־prepared statements הקיימת ברוב מסדי הנתונים הרלציונים, ולכן על כל אפליקציה לדאוג לתשתית ווידוא הקלט בעצמה.

לסיכום:

סודיות אין תמיכה בהצפנת קבצי המידע של מסד הנתונים, ובגרסה הזמינה העיקרית של מסד הנתונים אין תמיכה בהצפנת המידע המועבר בין מסד הנתונים לאפליקציות המתחברות. מנגנון ההרשאות של MongoDB הוא פשטני ביותר ומתאפשר רק ברמת מסד הנתונים כולו, ומאפשר או הרשאות קריאה בלבד, או קריאה־כתיבה לכל המידע הנשמר ברמת מסד הנתונים כולו. השימוש בשפת JavaScript כשפת גישה אל הנתונים פותח את מסד הנתונים להתקפות הזרקה (JavaScript Injection) המחייבות נקיטת משנה זהירות כאשר מפתחים אפליקציות מעל מסד נתונים זה.

שלמות אין מנגנון המוודא את שלמות הנתונים במסד הנתונים.

זמינות מסד הנתונים מאפשר ביזור של המידע מעל צבירי שכפול וצבירי פיזור כך שניתן להגיע לתצורה בה לפחות שרת אחד המכיל את המידע הנדרש יהיה תמיד זמין.

אימות מסד הנתונים מספק מנגנון אימות משתמשים באמצעות סיסמה. מנגנון האימות איננו מאוד חזק וניתן לתקוף אותו באמצעות התקפת מילון.

PostgreSQL 5.4

מסד נתונים זה היינו מסד נתונים רלציוני שהחל את דרכו באקדמיה בשנת 1986 כפרויקט שנועד להדגים טכנולוגיות הקשורות לתחום מסדי הנתונים וכפרויקט המשך למסד הנתונים Ingres (ולכן נקרא פרויקט Postgres). Postgres תוחזק ע"י Michael Stonebraker וסטודנטים שלו במשך 8 שנים, עד שנת 1994 [31]. במשך זמן זה, הפרויקט ניסה להציג מימוש לדוגמה של מסד נתונים מבוסס אובייקטים. בפרק זמן זה נוספו למסד הנתונים יכולות רבות, כגון שפות תכנות מובנות, אינדקסים מסוגים שונים, טיפוסים מבוססי הקשר, תכונות מונחות עצמים ועוד. בשנת 1995 כתבו שני תלמידי Ph.D של Stonebraker מימוש של שפת SQL למסד הנתונים, ושינו את שם הפרויקט ל־Postgre95. פרויקט זה שימש הבסיס לפרויקט PostgreSQL המודרני, שהחל משנת 1996 מפותח ע"י קהילת הקוד הפתוח תחת רישיון BSD. מאז ועד היום, PostgreSQL התקדם לגרסה 9.2, ומשמש מספר גדול של משתמשים, וביניהם ארגונים גדולים בתעשייה כגון Cisco, RIM, Sony ו־NASA [32]. ל־PostgreSQL תכונות ויכולות רבות המעמידות אותו בקנה אחד עם מיטב מסדי הנתונים הרלציונים המסחריים, כולל יכולות מרשימות באבטחת הנתונים הנשמרים בתוכו.

כדי לשמור על סודיות הנתונים, PostgreSQL מגיע עם תמיכה מובנית בפרוטוקול SSL עבור כל חיבור אל מסד הנתונים. יש תמיכה מלאה בכל היכולות של SSL, כולל בדיקת רשימות תעודות מבוטלות (Certificate Revocation Lists). יש למסד הנתונים תמיכה מובנת להפעלת הצפנה עבור הנתונים הנשמרים בתוכו דרך ספריית הרחבה בשם pgcrypto, המאפשרת שימוש בהצפנות סימטריות ואסימטריות, כולל תמיכה מלאה בגיבובים קריפטוגרפיים. כאשר משלבים בין שתי היכולות, המידע הנמצא במסד הנתונים גם עובר מוצפן על גבי הרשת, וגם נשמר מוצפן בקבצי מסד הנתונים, ברמת הטור הבודד. פתרון ההצפנה מבוסס pgcrypto מצפה לקבל בכל הפעלה את המפתח של ההצפנה, ולכן ניהול המפתחות נעשה באחריות האפליקציה המשתמשת ביכולות, ולא באחריות מסד הנתונים. PostgreSQL מאפשר כברירת מחדל ניהול הרשאות לפי מודל RBAC כאשר יש תמיכה ניסיונית [33] במודל הרשאות מנדטוריות מבוסס תוויות (label-based mandatory access control) כאשר מסד הנתונים מותקן על Linux (המימוש נעשה באמצעות SELinux של ה־NSA). יכולות אלה מאפשרות למשתמש להגדיר את ההרשאות הנדרשות על המידע לכל משתמש במערכת, ומאפשרות ל־PostgreSQL לספק את דרישת סודיות הנתונים.

PostgreSQL תומך באימות זהות המשתמשים המתחברים אל מסד הנתונים במגוון מנגנונים, כגון: משתמש וסיסמא, שימוש בפרוטוקול SSL עם יכולת זיהוי הלקוח (client authentication), שימוש במנגנונים של מערכת ההפעלה כגון PAM בפלטפורמות התומכות בכך, וכמובן גם ב־LDAP וב־Kerberos. בחירת מנגנון אימות זהות המשתמש היא אחריות מנהל מסד הנתונים, כאשר ברור

שחלק מהמנגנונים דורשים תשתית תומכת מתאימה ברמת מערכת ההפעלה והרשת בה מותקן מסד הנתונים.

לקבצי הנתונים של PostgreSQL אין מנגנון הגנה המסוגל לזהות שינוים בקבצי הנתונים שאירעו בזמן שמסד הנתונים היה מכובה. מסד הנתונים כן יזהה בזמן עליה אם קבצי הנתונים אינם תקינים וישנם מנגנונים המאפשרים להתמודד עם מצבים אלה. ניתן להשתמש במנגנון ה־trigger המובנה של PostgreSQL כדי לעקוב אחרי כל גישה למסד הנתונים ויצירת מעקב אחרי הנתונים (audit trail). מסד הנתונים תומך בכל בהגדרת referential constraints לווידוא קונסיסטנטיות של נתונים ו־integrity constraints לווידוא שלמות של נתונים ברמת מסד הנתונים עצמו.

PostgreSQL תומך ביכולות שכפול replication המאפשרים תמיכה בהתאוששות מתקלות בשיטת Warm Standby, לפיה כל פעולה המבוצעת במסד הנתונים נשמרת בקבצי תנועות, וקבצי התנועות האלה מבוצעים אח"כ בשרת אחד או יותר נוספים. בשיטה זאת, אם השרת הראשי איננו זמין, העבודה יכולה להמשך מול אחד מהשרתים האחרים, וללא הפסד של נתונים. בגרסאות האחרונות של מסד הנתונים, ניתן להשתמש בשרתים הנוספים הקיימים בשיטת Hot Standby, לפיה אפליקציה יכולה להתחבר אל שרת הנמצא במצב Standby ולבצע פעולות קריאה בלבד עליו, במקום על השרת הראשי. יכולות אלה מאפשרות למסד הנתונים לספק דרישות זמינות גבוהות ויכולת התאוששות מרשימה מתקלות.

ראה טבלה 2 לסיכום יכולות האבטחה של מסדי הנתונים הלא רלציונים הנסקרים בהשוואה ל־PostgreSQL.

6 מימוש מנגנון אימות והרשאות ב־Cassandra

כפי שהוצג למעלה, Cassandra איננו מספק מנגנון אימות ומנגנון הרשאות שניתן להפעיל ישירות. יש במסד הנתונים הכנה למנגנונים כאלה, ועל מנהל מסד הנתונים לספק את מימוש המנגנונים עצמם. בחלק זה אציג את מנגנוני האימות וההרשאות שמימשתי עבור Cassandra.

6.1 מנגנון ההזדהות

מרחב המשתמשים ב־Kerberos מחולק באופן טבעי לאזורי גישה (Realms). מנגנון האימות שמומש מאפשר ל־Cassandra להשתמש בשרת Kerberos לביצוע פעולת אימות המשתמשים. הנחת המוצא בפתרון הוא שכל משתמש המסוגל להציג תעודה (ticket) תקינה משרת ה־KDC עבור אזור הגישה (realm) אליו שייך מסד הנתונים הוא משתמש חוקי שמותרת לו גישה למסד הנתונים. משתמש המוגדר ב־Kerberos יכול להיות או משתמש (User) או שירות הניתן למשתמש (Service). המוסכמה בהגדרת שמות משתמשים ב־Kerberos היא

נקודת בדיקה		Cassandra	HBase	MongoDB	PostgreSQL
סודיות	אבטחת קבצים (הצפנה)	אין	אין	אין	תבילת pgcrypto
	אבטחת תעבורה	SSL בין חברי אשכול	תעבורה מוצפנת במנגנון ה־RPC	אין בגרסה הסטנדרטית, ניתן להדר גרסה מיוחדת	תמיכה מלאה ב־SSL
	מנגנון הרשאות	קיימת תשתית בקוד, אין מימוש כברירת מחדל	ארבע הרשאות בסיסיות	הרשאות קריאה בלבד או קריאה כתיבה על מסד נתונים שלם	תמיכה מלאה ב־RBAC וניסיונית דרך SELinux
שלמות נתונים	שלמות קבצי מידע	אין	מנגנון Checksum על בלוקים ברמת Hadoop	אין	אין
	שלמות מידע בתנועה	באמצעות SSL עבור תעבורה בין חברי אשכול	מידע מועבר מוצפן בפרוטוקול ה־RPC של HBase	אין	תמיכה מלאה ב־SSL
	שלמות המידע באפליקציה	אין	אין	אין	Referential Constraints
זמינות	יתירות	תצורת אשכול כשתפקיד כל מחשב באשכול זהה, מידע נשמר ביותר משרת אחד, בהתאם להגדרות האפליקציה	תצורת אשכול, כל סוג שרת משוכפל מספר פעמים. כל עוד יש שרת אחד מכל סוג זמין, המידע זמין	אשכולי פיזור, שכפול וקונפיגורציה	Warm Standby, Hot Standby
	התאוששות	ניתן להגדיל ולהקטין את האשכול באופן דינמי כדי להתאושש מנפילת חברים באשכול	ניתן להגדיל ולהקטין את האשכול באופן דינמי כדי להתאושש מנפילת חברים באשכול	ניתן להגדיל ולהקטין את האשכול באופן דינמי כדי להתאושש מנפילת חברים באשכול	
אימות	אימות זהות המשתמש	אין מימוש, יש הכנה לתמיכה	שימוש בפרוטוקול Kerberos	מנגנון אימות באמצעות משתמש וסיסמה	תמיכה במנגנונים רבים באופן ישיר (סיסמה,SSL) ובאופן עקיף (זיהוי באמצעות מערכת Kerberos, ההפעלה, ועוד).

טבלה 2: יכולות האבטחה של מסדי הנתונים הלא רלציונים

שמות שירותים	שמות משתמשים
$\underbrace{\text{host}}_{service} / \underbrace{kdc.example.com}_{hostname} @ \underbrace{EXAMPLE.COM}_{realm}$	$\underbrace{lior}_{user} @ \underbrace{EXAMPLE.COM}_{realm}$
$\underbrace{\text{cassandra}}_{service} / \underbrace{cas1.example.com}_{hostname} @ \underbrace{EXAMPLE.COM}_{realm}$	$\underbrace{lior}_{user} / \underbrace{DBA}_{instance} @ \underbrace{EXAMPLE.COM}_{realm}$

איור 9: שמות משתמשים ב־Kerberos

שאם המשתמש מייצג שירות, אז חלקו הראשון של שם המשתמש יהיה שם השירות, לדוגמה cassandra ושמו השני יהיה שמו המלא של נותן השירות, לדוגמה database.example.com. אם מדובר במשתמש (User), מקובל שחלקו הראשון של שם המשתמש ייצג את המשתמש, לדוגמה lior וחלקו השני (האופציונלי) ייצג את התפקיד שניתן למשתמש זה ב־Kerberos, לדוגמה DBA. באיור 9 ישנם דוגמאות לשמות משתמשים ב־Kerberos המייצגים שירותים ומשתמשים.

פתרון אימות זהות המשתמשים שמומש עבור Cassandra מכוון לפרוטוקול ה־RPC המשמש את המשתמשים המתחברים אל מסד הנתונים. בעת ביצוע ההתחברות על המשתמש להציג תעודה תקפה משרת ה־KDC. על תעודה זאת להתאים לשרת המדויק אליו מנסה המשתמש להתחבר. בעת ההתחברות, הצד המתחבר שולח את התעודה שקיבל משרת ה־KDC אל שרת מסד הנתונים במקום משתמש וסיסמא. שרת מסד הנתונים מפענח את התעודה, מוודא שהיא תקפה וחוקית לפי פרוטוקול Kerberos, ומשתמש במידע הקיים בתוך התעודה כדי לזהות את המשתמש במערכת.

כדי לאפשר למסד הנתונים לבדוק את תקינות התעודה שהוצגה לו, על כל שרת באשכול להזדהות בזמן עליית השרת אל שרת ה־Kerberos. לאחר שזה בוצע יכול השרת לאשר את התעודות המוצגות לו ע"י האפליקציות המתחברות. כדי לאפשר למסד הנתונים להזדהות בפני שרת ה־Kerberos, שומרים את מפתח הזיהוי של כל שרת בקובץ keytab של תשתית ה־Kerberos. בשפת Java ישנו מימוש סטנדרטי בשם JAAS (Java Authentication and Authorization Services) עבור שרתים ולקוחות המאפשר לאפליקציה הכתובה בשפת Java להשתמש בספריות של מערכת ההפעלה כדי לבצע את פעולות האימות. המימוש שלי משתמש בתשתית זאת גם בצד מסד הנתונים וגם בצד הלקוח המתחבר באמצעות פרוטוקול Thrift של מסד הנתונים. בנוסף, השתמשתי בספריית Kerberos של שפת python כדי לאפשר לאפליקציית הדוגמה cqlsh התומכת בשפת CQL של Cassandra גם כן להתחבר עם זיהוי באמצעות Kerberos.

להלן מספר דוגמאות להתחברות אל Cassandra המשתמש ב־Kerberos לביצוע ההזדהות.

ניסיון ראשון מבוצע ללא תעודה תקינה:

```
cassandra@cass1:cassandra-1.1.6$ klist
klist: No credentials cache found (ticket cache FILE:/tmp/krb5cc_1000)
cassandra@cass1:cassandra-1.1.6$ ./bin/cqlsh -z=t -3 cass1.example.com
Traceback (most recent call last):
  File "./bin/cqlsh", line 2787, in <module>
   main(*read_options(sys.argv[1:], os.environ))
  File "./bin/cqlsh", line 2773, in main
    display_float_precision=options.float_precision)
  File "./bin/cqlsh", line 578, in __init
    user=username, password=password)
  File "cql-1.4.0/cql/connection.py", line 152, in connect
  File "cql-1.4.0/cql/connection.py", line 61, in __init__
kerberos.GSSError:
  (('Unspecified GSS failure.  Minor code may provide more information',
          851968),
  ("Credentials cache file '/tmp/krb5cc_1000' not found", -1765328189))
cassandra@cass1:cassandra-1.1.6$
```

ניסיון שני מבוצע כאשר יש למשתמש המתחבר תעודה תקינה

```
cassandra@cass1:cassandra-1.1.6$ kinit alice
Password for alice@EXAMPLE.COM:
cassandra@cass1:cassandra-1.1.6$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: alice@EXAMPLE.COM

Valid starting     Expires            Service principal
11/23/12 10:17:12  11/23/12 20:17:12  krbtgt/EXAMPLE.COM@EXAMPLE.COM
        renew until 11/24/12 10:17:10
cassandra@cass1:cassandra-1.1.6$ ./bin/cqlsh -z=true -3 cass1.example.com
 Connected to Kerberos Auth Cluster at cass1.example.com:9160.
[cqlsh 2.2.0 | Cassandra 1.1.6-SNAPSHOT | CQL spec 3.0.0
             | Thrift protocol 19.33.0]
Use HELP for help.
cqlsh>
```

6.2 מנגנון ההרשאות

מנגנון ההרשאות שהוספתי מאפשר להגדיר הרשאות על מרחבי מפתחות ועל משפחות טורים במסד הנתונים. לדוגמה, ניתן להגדיר הרשאת SELECT עבור משתמש lior@EXAMPLE.COM על משפחת הטורים Users. הפתרון מגדיר קבוצה אחת מיוחדת הנקראת DBA, ומשתמשים המכילים בשמם את התפקיד הזה יקבלו הרשאות מלאות לכל מסד הנתונים. לדוגמה, המשתמש user/DBA@EXAMPLE.COM יהיה משתמש מורשה בעל הרשאות מלאות. מנגנון ההרשאות מגן על שלמות מסד הנתונים ע"י מניעת כל הרשאת כתיבה על מרחב המפתחות system, כך שרק מסד הנתונים עצמו יוכל לעדכן את המידע השמור במרחב מפתחות זה. ההרשאות עצמן נשמרות במסד הנתונים כחלק מהסכמה של מסד הנתונים, ולכן הן תופסות בכל הצביר מיד אחרי שניתנו או הורדו, ואין צורך לסנכרן ידנית את ההרשאות בתוך הצביר.

ההרשאות המשויכות למשאבים במסד הנתונים יצוינו כמסמך JSON, המכיל מפתחות בהם שם המפתח הוא שם המשתמש או הקבוצה המקבלת את ההרשאה, וערך המפתח הוא רשימת ההרשאות הניתנות למשתמש זה. לדוגמה, המסמך הבא מקנה הרשאות עדכון בלבד למשתמש user1@EXAMPLE.COM, והרשאות קריאה כתיבה לחברי הקבוצה read:

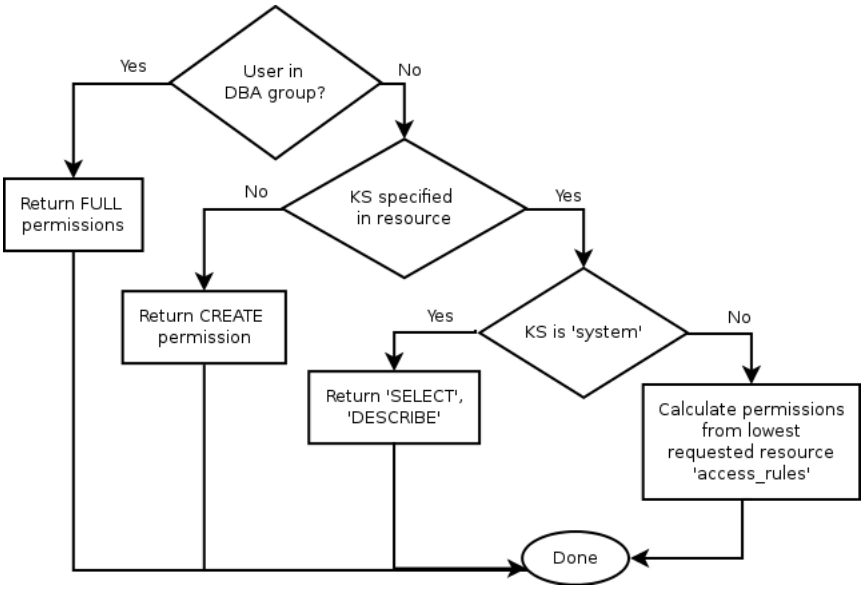
$$\{"user1@EXAMPLE.COM" : ["UPDATE"], "read" : ["SELECT","DESCRIBE"]\}$$

ראה איור 10 לתיאור מנגנון חישוב ההרשאות הניתנות למשתמש.

רשימת ההרשאות ששויכו למרחבי מפתחות ומשפחות טורים נשמרים כתור במשפחות הטורים בסכמת system של מסד הנתונים, ולכן הן מסונכרנות באופן מיידי ע"י מסד הנתונים לכל חברי האשכול. תמיכה הוספה גם למימוש של שפת CQL כדי לתמוך במנגנון ההרשאות המובנה לתוך סכמת מסד הנתונים, וכדי לאפשר להגדיר את ההרשאות בשפת מסד הנתונים.

בדוגמה הבאה ניתן לראות את הטור access_rules שהתווסף לטבלאות המערכת בסכמת system של Cassandra:

```
cassandra@cass1:cassandra-1.1.6$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: lior@EXAMPLE.COM
cassandra@cass1:cassandra-1.1.6$ ./bin/cassandra-cli -kr -j conf/jaas.conf \
  -h cass2.example.com
Connected to: "Kerberos Auth Cluster" on cass2.example.com/9160
Welcome to Cassandra CLI version 1.1.6-SNAPSHOT
Type 'help;' or '?' for help. Type 'quit;' or 'exit;' to quit.
```



איור 10: מנגנון חישוב הרשאות למשתמש

```
[lior@unknown] use system;
Authenticated to keyspace: system
[lior@system] list schema_keyspaces ;
Using default limit of 100
Using default column limit of 100
-------------------
RowKey: example
=>(column=access_rules,
        value={"cass@EXAMPLE.COM":["DESCRIBE"],
                "lior@EXAMPLE.COM":["DESCRIBE","CREATE",
                                    "ALTER","DROP"]},
        timestamp=1351840919169000)
=> (column=durable_writes, value=true, timestamp=1351840919169000)
=> (column=name, value=example, timestamp=1351840919169000)
=> (column=strategy_class,
        value=org.apache.cassandra.locator.SimpleStrategy,
        timestamp=1351840919169000)
=> (column=strategy_options,
        value={"replication_factor":"2"}, timestamp=1351840919169000)
1 Row Returned.
Elapsed time: 45 msec(s).
[lior@system] use example;
```

```
Authenticated to keyspace: example
[lior@example] describe cf1;
    ColumnFamily: cf1
      Key Validation Class: org.apache.cassandra.db.marshal.AsciiType
      Default column value validator: org.apache.cassandra.db.marshal.UTF8Type
      Columns sorted by: org.apache.cassandra.db.marshal.UTF8Type
      GC grace seconds: 864000
      Compaction min/max thresholds: 4/32
      Read repair chance: 0.1
      DC Local Read repair chance: 0.0
      Replicate on write: true
      Caching: KEYS_ONLY
      Bloom Filter FP chance: default
      Access List: {"lior@EXAMPLE.COM":["DESCRIBE","CREATE", "ALTER",
                                        "DROP","UPDATE","SELECT"],
                    "alice@EXAMPLE.COM":["UPDATE"]}
      Built indexes: []
      Column Metadata:
        Column Name: col2
          Validation Class: org.apache.cassandra.db.marshal.AsciiType
        Column Name: col1
          Validation Class: org.apache.cassandra.db.marshal.AsciiType
      Compaction Strategy:
         org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy
      Compression Options:
        sstable_compression: org.apache.cassandra.io.compress.SnappyCompressor
```

בדוגמה הבאה ניתן לראות שהמשתמש bob@EXAMPLE.COM איננו יכול לשלוף נתונים מ־example.cf1:

```
cassandra@cass1:cassandra-1.1.6$ kinit bob
Password for bob@EXAMPLE.COM:
cassandra@cass1:cassandra-1.1.6$ ./bin/cqlsh -z=t -3 cass2.example.com
Connected to Kerberos Auth Cluster at cass2.example.com:9160.
[cqlsh 2.2.0 | Cassandra 1.1.6-SNAPSHOT | CQL spec 3.0.0
           | Thrift protocol 19.33.0]
Use HELP for help.
cqlsh> use example;
```

```
cqlsh:example> select * from cf1;
Bad Request: #<User bob@EXAMPLE.COM groups=[]> does not have
    permission SELECT for /cassandra/keyspaces/example/cf1
Perhaps you meant to use CQL 2? Try using the -2 option when starting cqlsh.
cqlsh:example>
```

7 סיכום והצעות לכיווני מחקר נוספים

מבין מסדי הנתונים הלא רלציונים שנסקרו, ניכר שנושא האבטחה איננו הדגש העיקרי בעיצוב מערכות מסדי הנתונים האלה. יכולות האבטחה של MongoDB הן בסיסיות מאוד וקלות מאוד לעקיפה, ב־Cassandra ישנה רק תשתית המהווה הכנה למנגנון אבטחה אך ללא מימוש סטנדרטי במסד הנתונים, ורק ב־HBase יש מנגנון אבטחה המתקרב ביכולותיו למנגנוני האבטחה הקיימים בעולם מסדי הנתונים הרלציונים. באף אחד ממסדי הנתונים הלא רלציונים שנסקרו אין התייחסות רצינית לבעיית שלמות הנתונים. קל לראות שבהשוואה ליכולות של PostgreSQL, למסדי הנתונים הלא רלציונים יש עוד מרחק גדול מאוד להתקדם.

ישנם מספר כיוונים הבודקים כיצד ניתן לשלב הצפנה של קבצי המידע ברמת מערכת הקבצים המבוזרת HDFS המסופקת עם Hadoop [34, 35], ובנוסף הוספת יכולות הצפנה לפרוטוקול ה־RPC של Hadoop. כל גרסה חדשה של MongoDB משפרת מעט את המצב העגום של יכולות האבטחה במסד נתונים זה. הגרסה שנסקרה בעבודה זאת היא הראשונה שהכילה תמיכה באימות משתמשים בצביר, ותכונות אבטחה חדשות נמצאות בפיתוח וניתן לראות את תיאורי התכונות שעליהם עובדים המפתחים לקראת הגרסאות הבאות במערכת ניהול המוצר של הפרויקט. פרויקט Cassandra שילב כבר את התשתית ליכולות אימות וזיהוי וכפי שהדגמתי, כבר היום ניתן יחסית בקלות להוסיף קוד למסד הנתונים שיממש אימות וזיהוי איכותיים ברמת משפחות טורים. באופן כללי, נראה שמפתחי מסדי הנתונים הלא רלציונים משלבים יכולות אבטחה לתוך המוצרים. על מחקרים עתידיים לבדוק כיצד ניתן לשלב לתוך מסד נתונים שמבנהו איננו ידוע מראש יכולת להגן על הנתונים ולשמור על שלמותם. בעולם מסדי הנתונים הרלציונים יש למסד הנתונים את היתרון שמסד הנתונים ומנהל מסד הנתונים יודעים מראש את מבנה הנתונים הנשמרים במסד הנתונים ולכן יכולים להגן על המידע ביתר קלות. במקרה בו אין למסד הנתונים ידע מוקדם על מבנה הנתונים נראה שאין מתודולוגיה המתאימה להגנה על שלמות הנתונים שמוכנסים לתוך מסד הנתונים.

References

[1] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[2] Apache Cassandra. [Online]. Available: http://wiki.apache.org/cassandra/

[3] L. George, *HBase: The Definitive Guide*. O'Reilly Media, 2011.

[4] MongoDB documentation. [Online]. Available: http://www.mongodb.org/display/DOCS/Home

[5] A. Agarwal, M. Slee, and M. Kwiatkowski, "Thrift: Scalable Cross-Language Services Implementation," Facebook, Tech. Rep., April 2007. [Online]. Available: http://incubator.apache.org/thrift/static/thrift-20070401.pdf

[6] Apache AVRO. [Online]. Available: http://avro.apache.org/

[7] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[9] E. Brewer, "Towards Robust Distributed Systems," in *Keynote*. Proceedings of PODC, July 2000.

[10] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, January 2008.

[11] Ramakrishnan and Gehrke, *Database Management Systems*, 3rd ed. McGraw Hill, 2003, ch. 21, pp. 692–709.

[12] R. Hasan, S. Myagmar, A. J. Lee, and W. Yurcik, "Toward a threat model for storage systems," in *Proceedings of the 2005 ACM workshop on Storage security and survivability*, ser. StorageSS '05. New York, NY, USA: ACM, 2005, pp. 94–102.

[13] Using Oracle Virtual Private Database to Control Data Access. [Online]. Available: http://docs.oracle.com/cd/B28359_01/network.111/b28531/vpd.htm#CIHCBCFJ

[14] (2011, November) Transparently Encrypted PostgreSQL Data Types. [Online]. Available: https://bitbucket.org/ivoras/pgenctypes/overview

[15] D. Doubrovkine. (2011, January) 2011 - the Year of NoSQL Data Breaches? [Online]. Available: http://www.teamshatter.com/uncategorized/2011-%E2%80%93-the-year-of-nosql-data-breaches/

[16] E. Chickowski. (2012, January) Does NoSQL Mean No Security? [Online]. Available: http://www.darkreading.com/database-security/167901020/security/news/232400214/does-nosql-mean-no-security.html

[17] A. Lane. (2012, February) A Response To NoSQL Security Concerns. [Online]. Available: http://www.darkreading.com/blog/232600288/a-response-to-nosql-security-concerns.html

[18] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, "Security Issues in NoSQL Databases," *IEEE TrustCom/IEEE ICESS/FCST, International Joint Conference of*, pp. 541–547, 2011.

[19] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, June 2008.

[20] K. Muthukkaruppan. (2010, November) The Underlying Technology of Messages. [Online]. Available: http://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919

[21] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, ser. STOC '97. New York, NY, USA: ACM, 1997, pp. 654–663.

[22] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/

[23] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, Oct. 2003.

[24] Apache Zookeeper. [Online]. Available: http://zookeeper.apache.org/

[25] Bson: Binary JSON. [Online]. Available: http://bsonspec.org/

[26] A. Freier, P. Karlton, and P. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0," RFC 6101 (Historic), Internet Engineering Task Force, Aug. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6101.txt

[27] B. C. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," *Comm. Mag.*, vol. 32, no. 9, pp. 33–38, Sep. 1994.

[28] (2011, August) DigiNotar reports security incident. [Online]. Available: http://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_diginotar_reports_security_incident.aspx

[29] Proxmox VE. [Online]. Available: http://pve.proxmox.com/wiki/Main_Page

[30] A. Melnikov and K. Zeilenga, "Simple Authentication and Security Layer (SASL)," RFC 4422, Internet Engineering Task Force, Jun. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4422.txt

[31] (2012, October) PostgreSQL History. [Online]. Available: http://www.postgresql.org/about/history/

[32] D. P. Duncavage. (2010, July) NASA needs Postgres - Nagios help. [Online]. Available: http://archives.postgresql.org/pgsql-general/2010-07/msg00394.php

[33] (2012, October) Security Enhanced PostgreSQL. [Online]. Available: http://www.postgresql.org/docs/9.2/interactive/sepgsql.html

[34] H.-Y. Lin, S.-T. Shen, W.-G. Tzeng, and B.-S. P. Lin, "Toward Data Confidentiality via Integrating Hybrid Encryption Schemes and Hadoop Distributed File System." in *AINA*, L. Barolli, T. Enokido, F. Xhafa, and M. Takizawa, Eds. IEEE, 2012, pp. 740–747.

[35] D. Doubrovkine. (2012, October) Hadoop Crypto Compressor. [Online]. Available: https://github.com/geisbruch/HadoopCryptoCompressor

נספחים

א' מימוש מנגנון אימות המשתמש ב־Cassandra

```
package name.okman.cassandra.auth;

import java.security.PrivilegedAction;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import org.apache.cassandra.auth.AuthenticatedUser;
import org.apache.cassandra.auth.IAuthenticator;
import org.apache.cassandra.config.ConfigurationException;
import org.apache.cassandra.config.DatabaseDescriptor;
import org.apache.cassandra.thrift.AuthenticationException;
import org.apache.commons.codec.binary.Base64;
import org.ietf.jgss.GSSContext;
import org.ietf.jgss.GSSCredential;
import org.ietf.jgss.GSSManager;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Implement an authenticator class based on Kerberos tickets.
 * @author Lior Okman
 */
public class KerberosAuth implements IAuthenticator {

  private static Logger logger = LoggerFactory.getLogger(KerberosAuth.class);

  /// This host's Kerberos subject, used to perform actions in an authenticated
  /// context
      private Subject subject;
```

```
    /// The name of the key that contains the ticket in the authentication
    /// request
    public static final String TICKET = "TICKET";

    public KerberosAuth() {
      System.setProperty( "javax.security.auth.useSubjectCredsOnly",
"true");
    }

    /**
     * This implementation requires users to authenticate, and returns
     * NULL as the default user.
     */
    @Override
    public AuthenticatedUser defaultUser() {
        return null;
    }

    /**
     *  Authenticate a user based on the provided credentials.
     *  @param credentials A map that should contain a single
     *                                  entry - TICKET that contains a valid
Kerberos ticket
     *  @return An AuthenticatedUser instance for the authenticated user
     *  @throws AuthenticationException Thrown if the authentication fails
     */
    @Override
   public AuthenticatedUser authenticate( Map<? extends CharSequence,
? extends CharSequence> credentials) throws AuthenticationException {
        if (credentials == null || credentials.isEmpty()) {
            logger.error("Didn't receive a ticket at all");
            return null;
        }
        if (!credentials.containsKey(TICKET)) {
           logger.error("Credentials not empty, but didn't contain
a ticket");
            return null;
        }
        ;
      final byte[] serviceTicket = Base64.decodeBase64(credentials.get("TICKET").toString().getBytes
```

```
            logger.debug("Got a ticket with "+ serviceTicket.length +"
bytes");
            String username = Subject.doAs( subject, new PrivilegedAc-
tion<String>() {
          public String run() {
            try {
              // Identify the server that communications are being made to.
              GSSManager manager = GSSManager.getInstance();
           GSSContext context = manager.createContext( (GSSCredential)
null);
          context.acceptSecContext( serviceTicket, 0, serviceTicket.length);
              return context.getSrcName().toString();
            }
            catch ( Exception e) {
          logger.error("Exception while authenticating a connection",
e);
              return null;
            }
          }
        });
      if (username == null) {
            throw new AuthenticationException("Failed to validate
the provided kerberos ticket");
      }
      logger.debug(username +" successfully authenticated");
      String[] userAndRealm = username.split("@");
      Set<String> groups = new HashSet<String>();
      String[] components = userAndRealm[0].split("/");
      for (int i=1; i<components.length; ++i) {
            groups.add(components[i]);
      }
      username = components[0] + "@" + userAndRealm[1];
      logger.debug("Authenticated as "+username+" with groups: "
+ groups);
          return new AuthenticatedUser(username, groups);
    }

    /**
     * Login to the KDC using JAAS and keep an authenticated subject.
     */
```

```
    @Override
    public void validateConfiguration() throws ConfigurationException
{
        System.setProperty( "java.security.auth.login.config",
                DatabaseDescriptor.getKerbAuthenticatorJaasFile());
      LoginContext loginCtx = null;
      try {
            loginCtx = new LoginContext( "Server" ) ;
              loginCtx.login();
        } catch (LoginException e) {
              throw new ConfigurationException("Failed to login
to Kerberos", e);
        }
      this.subject = loginCtx.getSubject();
    logger.info("Authenticated to Kerberos as '"+ subject.getPrincipals()
+"'");
    }

}
```

ב' מימוש מנגנון ההרשאות ב־Cassandra

```
package name.okman.cassandra.auth;

import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.Collections;
import java.util.EnumSet;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.cassandra.auth.AuthenticatedUser;
import org.apache.cassandra.auth.IAuthority2;
import org.apache.cassandra.auth.Permission;
import org.apache.cassandra.auth.Resources;
import org.apache.cassandra.config.CFMetaData;
import org.apache.cassandra.config.ConfigurationException;
```

```
import org.apache.cassandra.config.KSMetaData;
import org.apache.cassandra.config.Schema;
import org.apache.cassandra.cql3.CFName;
import org.apache.cassandra.thrift.Column;
import org.apache.cassandra.thrift.CqlMetadata;
import org.apache.cassandra.thrift.CqlResult;
import org.apache.cassandra.thrift.CqlResultType;
import org.apache.cassandra.thrift.CqlRow;
import org.apache.cassandra.thrift.InvalidRequestException;
import org.apache.cassandra.utils.ByteBufferUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Provide authorization for the provided user based on the authorization information
 * kept on the keyspace and the column family metadata respectively.
 *
 * @author Lior Okman
 */
public class Authorizor implements IAuthority2 {

  private static Logger logger = LoggerFactory.getLogger(Authorizor.class);
   /// The name of the super-user group.
   private static final String superUserGroup = "DBA";

      public Authorizor() {
      }

      @Override
      public void setup() {
      }

      /**
       * Called by Cassandra when the database needs to receive a set of
       * permissions that are available to the provided user on the
       * provided resource.
       * @param user The user whose permissions need to be provided
       * @param resource The resource that the user is attempting to access
      * @return An EnumSet<Permission> of the permissions available for
```

```
the
         *                    provided user on the provided resource.
         */
        @Override
    public EnumSet<Permission> authorize(AuthenticatedUser user, List<Object>
resource) {
            if (logger.isDebugEnabled())
                                logger.debug("Asked to retrieve permissions
for "+user.username + " on resource '" + Resources.toString(resource)
+ "'");

          // Permissions requested on the resource string "/cassandra/keyspaces"
without
            // a keyspace or column family
            if (resource.size() <= 2) {
                return Permission.ALL;
            }

            KSMetaData ksMetaData = null;
            CFMetaData cfMetaData = null;
            if (resource.size() >= 3) {
                // Only the keyspace
                ksMetaData = Schema.instance.getTableDefinition((String)
resource.get(2));
            }
            if (resource.size() >= 4 && ksMetaData != null) {
                // Also the column family
              cfMetaData = ksMetaData.cfMetaData().get(resource.get(3));
            }

            // A superuser should be able to do anything in the database.
            if (user.groups.contains(superUserGroup)) {
                return Permission.ALL;
            }

            if (ksMetaData == null) {
            EnumSet<Permission> createOnly = EnumSet.noneOf(Permission.class);
                createOnly.add(Permission.CREATE);
                return createOnly;
            }
```

```
            if (ksMetaData.name.equals("system"))
            {
            EnumSet<Permission> readOnly = EnumSet.noneOf(Permission.class);
                    readOnly.add(Permission.SELECT);
                    readOnly.add(Permission.DESCRIBE);
                    return readOnly;
            }

          // Calculate the full set of permissions using the lowest available level.
        EnumSet<Permission> resultSet = EnumSet.noneOf(Permission.class);
            if (cfMetaData != null) {
                    for (String group : user.groups) {
                            if (cfMetaData.accessList.containsKey(group)) {
                             resultSet.addAll(cfMetaData.accessList.get(group));
                            }
                    }
                    if (cfMetaData.accessList.containsKey(user.username)) {
                   resultSet.addAll(cfMetaData.accessList.get(user.username));
                    }
            }
            else if (ksMetaData != null) {
                    for (String group : user.groups) {
                            if (ksMetaData.accessList.containsKey(group)) {
                             resultSet.addAll(ksMetaData.accessList.get(group));
                            }
                    }
                    if (ksMetaData.accessList.containsKey(user.username)) {
                  resultSet.addAll(ksMetaData.accessList.get(user.username));
                    }
            }
            return resultSet;
    }

    @Override
    public void validateConfiguration() throws ConfigurationException
{
    }
```

```
    @Override
     public void grant(AuthenticatedUser granter, Permission permission,
String to, CFName resource, boolean grantOption) throws InvalidRequestEx-
ception {
          logger.debug(granter+" asked to grant "+permission+" to "+
to +" on "+resource);
      CFMetaData cfMetaData = Schema.instance.getCFMetaData(resource.getKeyspace(),
resource.getColumnFamily());
          EnumSet<Permission> permissions = null;
          if (cfMetaData.accessList.containsKey(to)) {
              permissions = cfMetaData.accessList.get(to);
          } else {
              permissions = EnumSet.noneOf(Permission.class);
          }
          permissions.add(permission);
          cfMetaData.accessList.put(to, permissions);
               logger.debug("Permissions enumset is now: " + cfMeta-
Data.accessList );
    }

    @Override
     public void revoke(AuthenticatedUser revoker, Permission permission,
String from, CFName resource) throws InvalidRequestException {
          logger.debug(revoker+" asked to revoke "+permission+" from
"+ from +" on "+resource);
      CFMetaData cfMetaData = Schema.instance.getCFMetaData(resource.getKeyspace(),
resource.getColumnFamily());
          EnumSet<Permission> permissions = null;
          if (cfMetaData.accessList.containsKey(from)) {
              permissions = cfMetaData.accessList.get(from);
              if (permissions.contains(permission)) {
                  permissions.remove(permission);
                  if (permissions.isEmpty()) {
                      cfMetaData.accessList.remove(from);
                  }
              }
          }
               logger.debug("Permissions enumset is now: " + cfMeta-
Data.accessList );
    }
```

```
    @Override
    public CqlResult listPermissions(String username) throws InvalidReques-
tException {
            CqlResult res = new CqlResult(CqlResultType.ROWS);
                res.schema = new CqlMetadata(Collections.<ByteBuffer,
String>emptyMap(),
                        Collections.<ByteBuffer, String>emptyMap(),
                        "AsciiType", "AsciiType");

            ByteBuffer keyspaceBytes = ByteBufferUtil.bytes("keyspace");
              ByteBuffer columnfamilyBytes = ByteBufferUtil.bytes("column
family");
            ByteBuffer typeBytes = ByteBufferUtil.bytes("type");
            ByteBuffer nameBytes = ByteBufferUtil.bytes("name");
            ByteBuffer permsBytes = ByteBufferUtil.bytes("permissions");
            List<CqlRow> rows = new ArrayList<CqlRow>();

          for (KSMetaData currKS : Schema.instance.getTableDefinitions())
{
                  if (currKS.accessList.containsKey(username)) {
                      List<Column> columns = new ArrayList<Column>(3);
                        columns.add(new Column(nameBytes).setValue(
                                ByteBufferUtil.bytes(currKS.name)));
                  columns.add(new Column(typeBytes).setValue(keyspaceBytes));
                        columns.add(new Column(permsBytes).setValue(
                        ByteBufferUtil.bytes(currKS.accessList.get(username).toString())));
                  rows.add(new CqlRow(ByteBufferUtil.bytes(currKS.name),
columns));
                  }
                        for (Map.Entry<String, CFMetaData> currCF : cur-
rKS.cfMetaData().entrySet()) {
                   if (currCF.getValue().accessList.containsKey(username))
{
                       List<Column> columns = new ArrayList<Column>(3);
                           columns.add(new Column(nameBytes).setValue(
                             ByteBufferUtil.bytes(currKS.name+"."+currCF.getKey())));
                     columns.add(new Column(typeBytes).setValue(columnfamilyBytes));
                           columns.add(new Column(permsBytes).setValue(
                             ByteBufferUtil.bytes(currCF.getValue().accessList.get(username).toString()
```

```
                    rows.add(new CqlRow(ByteBufferUtil.bytes(currKS.name+"."+currCF.getKey()),
columns));
                        }
                    }
                }
                res.setRows(rows);
                return res;
        }

}
```

Contents

1 Abstract

In July 2000, at a Parallel Computing conference, Eric Brewer posited that in asynchronous distributed systems only two of the following attributes can be achieved: Persistency, Availability, or Consistency. Relational database systems used in distributed settings that provide ACID guarantees were identified as architectural components that do not guarantee the Availability attribute, and because of that, can degrade the user experience. Consequently, non-relational database systems started appearing on the Internet, designed so that the developers using these database systems could choose which of the two attributes they preferred to guarantee, based on the system being developed. These non-relational database systems were designed to implement only a subset of the functionality provided by relational database systems, in order to allow for better performance, or distributed capabilities. For example, these non-relational database systems do not support the SQL language, and usually do not require the application developer to define the data model to be stored in the database. Since the original purpose for these database systems was to assist in the development of Internet sites and applications, the technology used to develop them drew heavily from the technology used to develop web sites. JSON was used to describe data, JavaScript and similar programming languages were used to query the database. Due to the lack of support for SQL, these database systems were nicknamed NoSQL databases. Most of these databases also enabled the application to utilize MapReduce algorithms, in order to query these distributed databases efficiently.

Security in relational databases can be described using four basic requirements in a model called CIAA:

1. Confidentiality: A non-privileged user must not be allowed to access data.
2. Integrity: Only authorized users can update data.
3. Availability: A user can always execute any operation for which he has permissions.
4. Authentication: Each user's identity can be verified.

In order to implement these requirements, relational databases define a rich Access Control Language, allowing privileges to be assigned to database users. For example, it is possible to grant a specific user the privilege to insert data into a table, while at the same time preventing any other operation to be performed

on that table. This is possible because the relational database has an accurate data model and the database engine rigorously enforces this model. In Oracle's RDBMS, permissions can be defined at an even lower level. Using the Virtual Private Database capabilities available from version 8i of the software, it is possible for the administrator to define policies that dynamically rewrite each query as it is executed in order to assert a predefined security policy. This policy can control security settings on a row-by-row level in each table or view. Relational databases can protect the information stored in them at the disk level, or even in specific columns in specific tables (e.g., in PostgreSQL). Relational databases also protect data while in transit by protecting the protocols used to transmit this data to database clients. Authentication in relational databases can be done in tandem with external services, such as Kerberos and Active Directory, allowing the database to be absolutely certain of the connecting user's identity.

During the initial explosion of NoSQL technologies, security capabilities were not given a very high priority. Since NoSQL databases do not usually have a catalog defining the data model, providing and maintaining access controls on the data is inherently more difficult in these systems. Despite the fact that these database systems are usually highly distributed, little thought was given to protecting data while it is transmitted between different parts of the database system's architecture. Most of these systems do not support data encryption on the data file level, and are vulnerable to attack from their host operating system and network. While assessing the security capabilities of MongoDB and Cassandra, multiple vulnerabilities were discovered. These allow attackers to bypass the database's access control systems and access data without privileges, or even to affect the database's availability.

In today's Internet, NoSQL databases are used by large organizations, such as Google and Facebook, in order to implement parts of their portfolios. Because of the ability of these NoSQL databases to scale to very large sizes, the security capabilities of these databases are becoming more and more important, as more and more companies use these systems to build their products.

In this work, I review three non-relational databases (Cassandra, HBase and MongoDB) based on existing publications and my personal experience gained by installing and studying these products. I was the lead author of a paper called "Security Issues in NoSQL Databases" (published at the TrustCom 2011 convention), which studied the security capabilities of Cassandra and MongoDB. This work extends that article by additionally reviewing the HBase database, and by

updating the first article's conclusions to cover later versions of Cassandra and MongoDB.

The Open University

Department of Math and Computer Science

Reviewing Security in Non-Relational Databases

Paper submitted as partial fulfillment of the requirements

towards a Master of Science in Computer Science

The Open University

Department of Math and Computer Science

November 2012

By: Lior Okman
Prepared under the supervision of Prof. Ehud Gudes