סיכום הרצאה – כלים עיקריים בגוגל

http://www.youtube.com/watch?v=oRwFpQKgRps&feature=related

גוגל מסתמכים על חומרה זולה יחסית, אך מפצים על כך בכמויות גדולות ובכתיבת תוכנה סלחנית כלפ תקלות חומרה.
המידע איתו מתמודדים הוא מידע בנפח עצום, ולכן חישובים חייבים להיות יעילים ומהירים.
מהסיבה הזו, משתמשים בגוגל בכלים מבוזרים GFS, MapReduce ו-BigTable.

החומרה

גוגל מאכסנים את המחשבים שלהם בData Centers ייעודיים. המחשבים מאוחסנים בקבוצות המכונות Rack, כאשר התקשורת הפנימית בכלRack מהירה יותר מתקשורת חיצונית המחשבים בנויים מחומרה זולה יחסית, ומריצים גרסה מותאמת של לינוקס.

מערכת הקבצים – GFS

מערכת הקבצים בה משתמשיהיא GFS, מערכת קבצים מבוזרת שמותאמת לשימוש עם קבצים גדולים מאוד.
בבסיסה, מערכת הקבצים מבוססת על מחשב שליטה מרכזי– Master, ומספר גדול של מחשבים בהם מוחזק המידע עצמו – Chunk Servers.
הקבצים עצמם מחולקים לחתיכות גדולות יחסית– 64 מגה כל אחת. כל חתיכה כזו נמצאת על יותר ממחשב אחד (בדרך כלל שלושה), כאשר ה-Master יודע היכן ניתן למצוא כל חתיכה.
כאשר לקוח מבקש לאחזר קובץ– הוא פונה אל ה-Master ומקבל את המיקומים המתאימים עבור חתיכות הקובץ הדרושות לו. לאחר מכן, פונה הלקוח בעצמו אל המחשבים שמכילים את הפריטים הדרושים ומקבל את המידע באופן ישיר. בצורה זו ניתן לטפל בכמויות גדולות מאוד של מידע ובקשות בו זמנית.

אלגוריתם מבוזר – MapReduce

כדי לבצע פעולות בצורה מבוזרת משתמשים בגוגל בצורה מאוד רחבה באלגוריתם בשם MapReduce, שמאפשר לחלק חישוב על גבי אוסף גדול של מחשבים
פעולת האלגוריתם מחולקת לשני שלבים עיקריים:
Map – בשלב זה עוברים על המידע ומאחזרים משם את כל הדרוש לנו בצורת זוגות של מפתח-ערך.
לדוגמה, אם אנו מעוניינים לספור מופעים של כל המילים באוסף גדול של מסמכים, בשלב ה-Map נעבור על כל המסמכים ועבור כל מסמך ניצור אוסף זוגות בהם עבור כל מילה במסמך יופיע הערך 1.
Reduce – בשלב זה עוברים על כל הרשומות שיצרנובשלב ה-Map ומשם מחלצים את התשובה לשאלה.
בדוגמה שלנו, עוברים על כל הזוגות ("מילה", 1) וממזגים אותם– הופכים שני מופעים מהצורה ("מילה", 1), ("מילה", 1) ל-("מילה", 2) וכן הלאה.
היתרון הגדול – ניתן לחלק הן את פעולת ה-Map והן את פעולת ה-Reduce בין מספר גדול של מחשבים.
אפשר לפתור מגוון מפתיע של בעיות בעזרת זה, החל מיצירת Inverted Index וכלה בדברים כמו תרגום מכונה.
בגוגל קיימת ספריה שממממשת את האלגוריתם בעזרתה ניתן לממש בקלות חישובים שדורשים כוח חישובי גדול.
הארכיטקטורה עצמה מבוססת על מחשבMaster שמנהל את העבודה, ומחשבים שמבצעים את העבודה.
כאשר לקוח מבקש לבצע עבודה כלשהי; ה-Master מחלק את העבודה לתתי-משימות ושולח למחשבים הזמינים, כאשר חלוקת העבודה יכולה להשתנות בצורה דינמית, אם למשל אחד המחשבים איטי במיוחד או הולך לעולמו במהלך העבודה– ה-Master יכול לחלק את המשימות שלו בין המחשבים האחרים או פשוט לתת לו פחות משימות מאשר לאחרים
לאחר ביצוע משימתMap, המידע נשמר מקומית על המחשבים שביצעו את הMap. לאחר שכל המחשבים סיימו לבצע את ה-Map, ה-Master מנהל את פעולת ה-Reduce – שולח את העבודה למחשבים שמיועדים

לכך. לא ניתן להתחיל עם ה-Reduce לפני סיום העבודה על ה-Map, כיוון שהספרייה מתחייבת ללקוח שעבור על מפתח נתון, בשלב ה-Reduce יתקבלו כל הערכים עבורו.

הספרייה של גוגל משתמשת בכל מיני שיטות לשיפור הביצועים

- עבודה מקומית – משתדלים לפעול בתוך אותו Rack (כיוון שהתקשורת הפנימית מהירה יותר מתקשורת בין Racks שונים)
- אפשר לקבוע בכמה מחשבים משתמשים אם רוצים חישוב מהיר יותר – אפשר להשתמש ביותר מחשבים.
- מאמץ רב לשיפור אלגוריתמי חיפוש ומיון – אפילו שיפור קטן בביצועים מביא לשיפור בביצועים של יישומים רבים שמסתמכים על הספרייה.
- שימוש ב-Pipeline – פעולת ה-Shuffle – חלוקת העבודה של ה-Reduce מתחילה להתבצע עוד במהלך שלב ה-Map.
- יצירת עותקים מקומיים של תוצאות ה-Map. בצורה זו, מחשבים מהירים יותר יכולים להתחיל פעולת Map חדשה בלי לחכות שה-Reduce של הפעולה הקודמת יסתיים.
- יצירה של יותר משימות ממחשבים זמינים במטרה להשיג חלוקת עומס טובה יותר.
- דחיסת מידע לפני שליחה.

מסד נתונים – BigTable

כלי שמאפשר פעולות מגוונות יותר מסתם פעולות על קבצים לא מדובר במסד נתונים קלאסי – אין תמיכה בפעולות בסיסיות כמו Join. לא מדובר גם ב-Hash Table מבוזר – יש תמיכה גם בפעולות יותר מתקדמות מחיפוש והכנסה. כמו עם שאר הכלים, גם כאן יש דרישה מובנית לכמויות אדירות של מידע, ולכן – ביזור שלו על פני מחשבים רבים.
באופן כללי, BigTable זו טבלה – מכילה עמודות ושורות. עם זאת, יש לה מימד נוסף – מימד הזמן. זה טוב למקרים בהם רוצים לשמור מגוון גרסאות (לדוגמה – תוכן של כתובת URL כלשהי, אשר עלול להשתנות עם הזמן).
הארכיטקטורה מבוססת על חלוקת הטבלה לאוסף של Tablets – עמודות (שלמות או חלקיות) בגודל של 100-200 מגה כל אחת. חלוקה זו לחלקים מאפשרת העברה מהירה של Tablet ממחשב למחשב, מה שנותן יכולות של ביזור וחלוקת עומס. כאמור, אפשר לחלק עמודה בטבלה ליותר מ-Tablet אחד, במידה וזו עמודה גדולה. גם הכיוון השני (מיזוג של Tablets) אפשרי, אבל קשה ומסובך יותר.
כעת ניתן להתבונן על המערכת בצורה רחבה יותר: השימוש ב-BigTable מתבסס על מערכת קבצים של GFS. גם כאן, קיים מחשב Master שמטרתו לנהל חלוקת עומס ולהחזיק את ה-Metadata. כלי בשם Cluster Scheduler אחראי על הקצאת משאבים וטיפול בתקלות (כאשר לקוח מעוניין לבצע כל מיני פעולות). מחשבים המכונים Tablet Servers מחזיקים את המידע עצמו. כלי מיוחד שנקרא Lock Service דואג לכך שיהיה Master אחד בכל פעם: רק מחשב שמצליח להשיג את המנעול יכול להיות Master. אם מחשב זה נופל, מחשב אחר יצליח להשיג את המנעול ולמלא את מקומו. בכל פעם יש כמה מחשבים שמנסים להשיג את המנעול (אחד פעיל, השאר מחכים למנעול) וכך מבטיחים רמה די גבוהה של אמינות בניגוד ל-GFS, כאן המידע על מיקומי המידע נמצא על המחשבים שמחזיקים את המידע ואין צורך לדבר קודם עם ה-Master ברוב המקרים. הגישה עצמה מתבצעת באמצעות ספרייה מיוחדת שמאפשרת שימוש נוח.

את המידע אפשר למצוא בעזרת טבלה מיוחדת שמאחסנת מצביעים למידע המסלול המלא של בקשת מידע מטבלה כולל גישה ראשונה ל-Lock Service שמביא מצביע לטבלה המכונה Meta0. מטבלה זו מקבלים מצביע לטבלה מסוג Meta1 (קיימות טבלאות רבות כאלו), כאשר טבלה זו מכילה מצביעים למידע (כלומר – ה-Tablet) הדרוש. אמנם מדובר במסלול שכולל שלוש גישות שונות, אבל המידע הזה בדרך כלל נשמר ב-Cache כך שלא צריך לבצע את כל המסלול כל פעם מחדש.
כל Tablet מכיל מידע לגבי שינויים שבוצעו עליו: קיים קובץ Log שמכיל את כל הפעולות האחרונות שבוצעו על כל Tablet, כאשר מדי פעם אפשר לאחד מידע על כמה שינויים בקובץ אחד.
ניתן לקבץ כמה Tablets שמכילים מידע שנפחו קטן (למשל – PageRank ו-Language) כדי להשיג גישה מהירה למידע משתי העמודות האלו בו זמנית, מבלי הצורך לחפש את שני ה-Tablets בנפרד.

בעיות קיימות

קיים קושי רציני לסנכרן ולחלוק מידע בין Clusters שונים באופן אוטומטי. בסך הכול, הסנכרון הפנימי בתוך אותו Cluster עובד מצוין, אבל עדיין לא קיימת דרך טובה לעבוד בקנה מידה גדול יותר(כלל עולמי).

דוגמאות שימוש

דוגמה מעניינת של שימוש באלגוריתמים המבוזרים שהוצגוהיא במנגנון של תרגום מכונה. בגוגל משתמשים במודל סטטיסטי המבוסס על לימוד של טקסטים מתורגמים קיימים שיפור משמעותי מושג בעזרת לימוד של שפת היעד, כך שהמשפטים שייווצרו יכילו ביטויים שנפוצים בשפת היעד(כדי שלא ליצור מבנה שאינו הגיוני בשפה). הכלים המבוזרים נדרשים מכיוון שמדובר בכמויות גדולות מאוד של מידע לעבד

סיכום הרצאה – ממשק משתמש בגוגל

http://www.youtube.com/watch?v=LT1UFZSbcxE&feature=related

עקומת הלימוד של חיפושים ברשת היא מהירה מאוד, כאשר משתמש ממוצע מגיע תוך כחודש לרמת מיומנות גבוהה. מסיבה זו החליטו בגוגל להתמקד במשתמשים מיומנים(למשל, לא להציג הרבה טקסטים של עזרה והכוונה), מבלי להפחיד משתמשים חדשים
חווית המשתמש בתוכנה מסוימת בדרך כלל מושפעת מהיכולות של התוכנה, כאשר משתמש מנוסה יותר בדרך כלל ידע להשתמש ביותר יכולות וחווית השימוש שלו תשתפר עם זאת, כאשר מעמידים בפני משתמש חדש מספר גחל של יכולות אפשר לקבל את האפקט ההפוך– המשתמש נרתע
בגוגל מתגברים על הבעיה הזו בעזרת הצגה של יכולות בהתאם למצב "הדברים שם כשאתה צריך אותם".
למשל: לא מציגים את יכולת התרגום בעמוד הראשי, אבל כאשר משתמש מקבל תוצאות בשפה אחרת מציגים את האפשרות לתרגם את המידע

ניסויים

קשה מאוד להחליט איך צריך לעצב דברים בשביל זה, פיתחו בגוגל טכניקה פסוודו-מתמטית: מציגים לכמה קבוצות משתמשים כמה אפשרויות שונות, ובודקים את תגובתם (נניח, האם זה גרם להם לבצע יותר חיפושים). כך אפשר להחליט באיזה ממשק לבחור(צורת הדף, צבעים וכן הלאה).
זה נותן תוצאות מדויקות יותר מאשר לשאול משתמשים, מכיוון שלא תמיד אנשים עונים בהתאם למה שהם רוצים.
דוגמה בולטת לכך היא השאלה– כמה תוצאות חיפוש כדאי להציג בכל דף, כאשר שאלו משתמשים הם ענו שכמה שיותר– עדיף. עם זאת, בקבוצת ניסיון שקיבלה דפים עם30 תוצאות ירד מספר החיפושים שהתבצעו בצורה דרמטית– משתמשים פשוט נטשו את האתר בסופו של דבר הסתבר שהזמן שארך לבצע את החיפוש כאשר מוצגות תוצאות רבות התארך ל0.9 שניות, וזה תסכל את המשתמשים

שיפור החיפוש

במקרים רבים המשתמש מנסה לחפש דברים בעזרת שאילתה לא טובה, או שגויה. כדי להשיג חיפוש טוב יותר צריך הבנה מסוימת של השאילתה למשל: כאשר משתמש מחפש שיר של להקה מסוימת עם שם לא נכון, אפשר לנחש שהוא מחפש שיר אחר עם שם דומה של אותה הלהקה, וזה מידע שאפשר להשיג מניתוח שאילתות רבות שמשתמשים אחרים ביצעו
כאשר משתמש מחפש "תצוגה מלמעלה של בריכה", הוא מתכוון פשוט ל"תמונות של בריכה", אבל לא כל כך קל לנחש את זה בצורה אוטומטית. דוגמאות כאלו ומסובכות יותר דורשות הבנה של השאילתה ובאופן כללי– מהוות בעיה קשה מאוד לפתרון.

גלובליזציה

הטמעה של מנגנון תרגום במנגנון החיפוש– אפשר לחפש בשפה אחת ולקבל תוצאות שמתורגמות לאותה השפה בזמן החיפוש (בעזרת תרגום מכונה). גרסה משוכללת יותר– חיפוש בכל השפות האפשריות ותרגום שמתבצע בצורה שקופה כך שבסופו של דבר התוצאות הן בשפת החיפוש, אבל המקורות יכולים להיות בכל מיני שפות.

חיפוש אוניברסאלי

רוצים לבצע חיפוש במספר מקורות מידע בו זמנית: לא רק באינדקס האתרים אלא גם בYouTube, בחדשות וכן הלאה. מתעוררת כאן בעיה – חיפוש כזה דורש שכל שאילתה היא למעשה אוסף של שאילתות כשכל אחת נשלחת למקור אחר. מגדילים את התנועה פי כמה אך רוצים להימנע מהצורך להכפיל את החומרה בהתאם.
בעיה נוספת היא סדר הדברים – מה להביא קודם, ואיך לסדר דברים שמגיעים ממקורות שונים (איך אפשר להשוות בין תפוזים לתפוחים?).

התאמה אישית

ניתן לחשב PageRank עבור קבוצות במקום עבור URL וכך לחסוך חישובים רבים (למשל: להסתכל על כל ה-URL ששייכים לאתר של האוניברסיטה כיחידה אחת). שיפור זה מאפשר לחשב ולהחזיק יותר מניקוד PageRank יחיד. למשל – PageRank נפרד עבור כל מדינה. הצעד הבא הוא PageRank עבור משתמש יחיד – בהתאם לחיפושים שבוצעו אפשר לתת ניקוד שונה לאתרים באופן אישי (Personalized Search).
כלי נוסף הוא iGoogle שמאפשר למשתמשים לחסיף יישומונים לאתר, שמאפשרים קיבוץ של מידע ממקורות שונים והצגתו בצורה מרוכזת בדף הבית
בשורה התחתונה – גוגל רוצים שמנוע החיפוש העתידי יכיר כל משתמש על סמך מידע שאפשר ללמוד מחיפושים קודמים, קישורים אליהם הוא נכנס, הצורה בה הוא ארגן את דף הבית שלו וכן הלאה