תיאור המערכת

ישויות במערכת

הגדרות

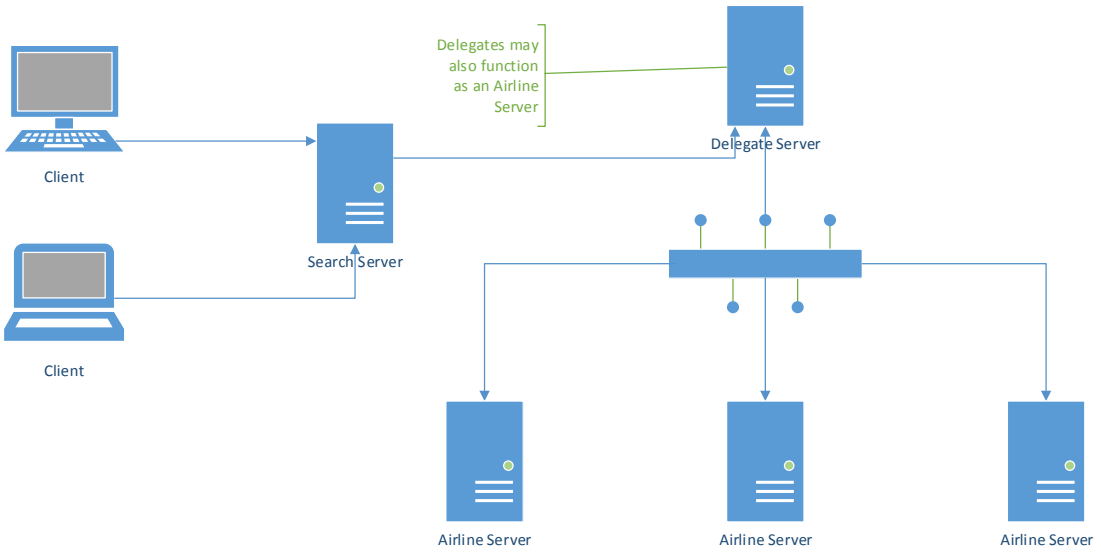
- לקוח – Client
- שרות חיפוש – FlightSearchServer
- מוכרן – AirlineServer
- שאילתה – בקשת query שהגיעה מלקוח
- קונקשן - טיסה שניה המתאימה לדרישות התרגיל שמוחזרת בשאילתה

תיאור

המערכת מורכבת מ-3 ישויות מרכזיות בדומה לתרגיל הקודם:

1. הלקוח שהינו צרכן המידע במערכת והמקור לגירויים.
2. שרת החיפוש (FlightSearchServer) שמפנה את הבקשות ליעד המתאים
3. שרות Alliance המאגד מספר מוכרני טיסות לישות אחת.

ה-Alliance מספק שרות פונקציונלי עבור הלקוח ע"י מציאת מסלול מתאים ממוצא כלשהו ליעד כלשהו דרך לכל היותר יעד שלישי (הטיסה השנייה הינה טיסת קונקשן) מכל המוכרנים השייכים ל-Alliance ושירות פונקציונלי ביחס לשרתי מוכרנים שהם חלק מה- Alliance והוא שרות רפליקציה העמיד בפני n-1 נפילות כאשר n הינו מספר המוכרנים המשתתפים. שרות הרפליקציה מאפשר נגישות למוכרנים אשר שרתיהם נפלו דרך שרתים של מוכרנים אחרים ובנוסף מאשר לבצע Load balancing של כמות המוכרנים למכונה (נשים לב כי פרמטר ה- Load balancing אינו לפי עומס בקשות אלא לפי מספר מוכרנים למכונה). כל Alliance חושף עצמו לשרות החיפוש ע"י delegate, לכל Alliance יש delegate יחיד.



ביזור מערכת ה-Alliance

בכדי למקסם את ה-Scalability המערכת אינה דורשת Leader כדי לבצע פעולות תחזוקה בהצטרפות\עזיבת מכונות. כל מכונה שמצטרפת מקבלת תמונת מצב ומתחילה להריץ אלגוריתם דטרמיניסטי כך שבכל שינוי במערכת כל המכונות רואות את אותה תמונת מצב ולכן מספיק שכל מכונה תעדכן רק את הרשומות שלה, דבר אשר חוסך מימוש נעילות באופן פרטני עבור מקרי קצה בעץ ה-ZooKeeper. המערכת הינה Plug-and-Play באופן מלא.

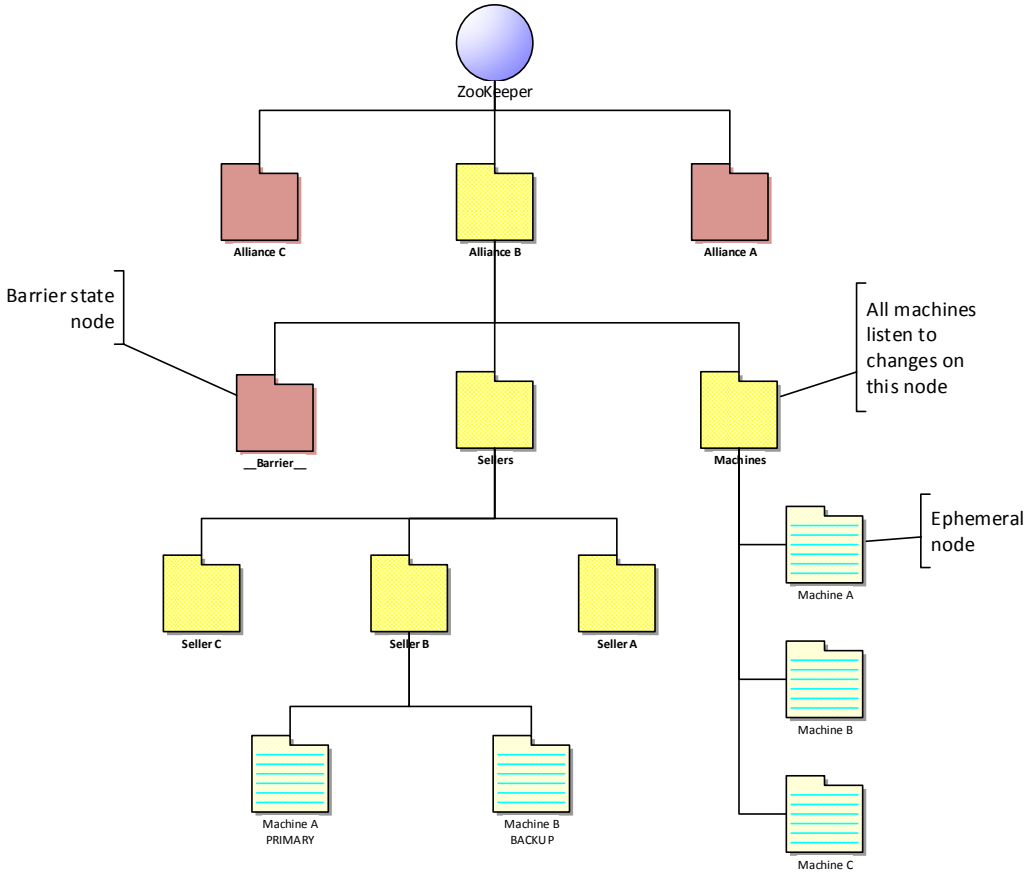
ה-Tradeoff בגישה זו הוא שהמימוש הופך למסובך מאוד. עוצמת החישוב הנדרשת זניחה יחסית ולכן לא נתחשב בה כגורם בעייתי.

הממשק מול -ZooKeeper

בכדי לזהות שינויים במערכת באופן קונסיסטנטי דרוש מנגנון קונצנזוס עם יכולת MEMBERSHIP. בתרגיל השתמשנו ב-ZooKeeper למטרה זו שכן המערכת מספקת ספריה מונחת אירועים וכן בנינו תשתית לניהול עץ לוקאלי המתעדכן בשינויים (Diff Merging) שמנוהלת ע"י מנגנון הרפליקציה.

המערכת מתוכננת כך שהיא מפרידה בין המידע הלוגי (מוכרן) לבית היחידה החישובית המריצה אותו (מכונה). באופן זה ניתן לבצע העברה יעילה של יחידות לוגיות בין מכונות שונות ולבצע ניטור נפילות ובחירת Delegate.

להלן תרשים המציג את היררכית העץ:



הצמתים הצהובים הם צמתים שכל ישות ב-Alliance מכירה (בהתאמה ל-Alliance בו היא נמצאת).

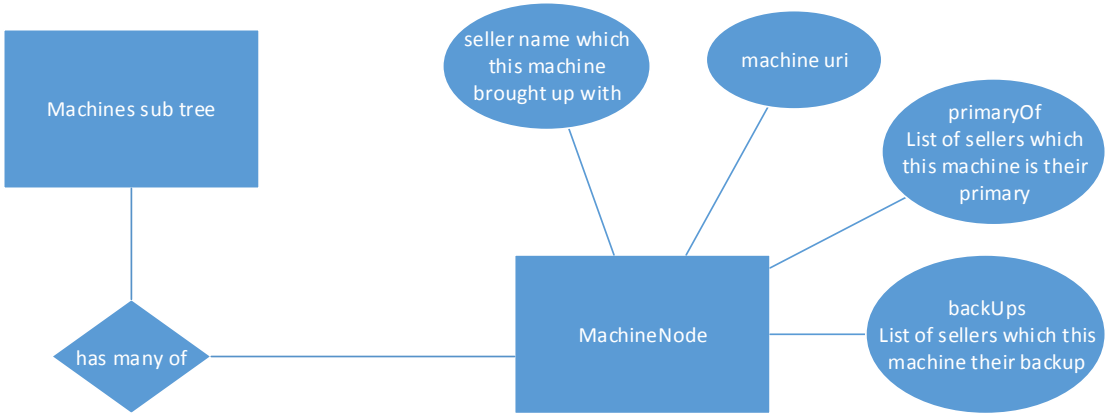
עבור כל Alliance קיים תת-עץ תחת שורש עץ ה ZooKeeper, (מעתה עץ ה-Alliance). תחת עץ זה רשומים כל המכונות והמוכרנים אותם ה-Alliance.

עבור כל מכונה שמצטרפת למערכת יוצרים עלה בענף המכונות (alliance/machines/) מסוג ephemeral כך שכל מי שנרשם לאירועי עדכונים על ענף זה יקבל אירוע עדכון במידה ומכונה הרשומה בענף זה כשלה כמו כן, אם לא קיים ענף עבור המוכרן אתו היא עלתה היא יוצרת עבורו ענף ורושמת עצמה כבן מסוג PRIMARY שלו.

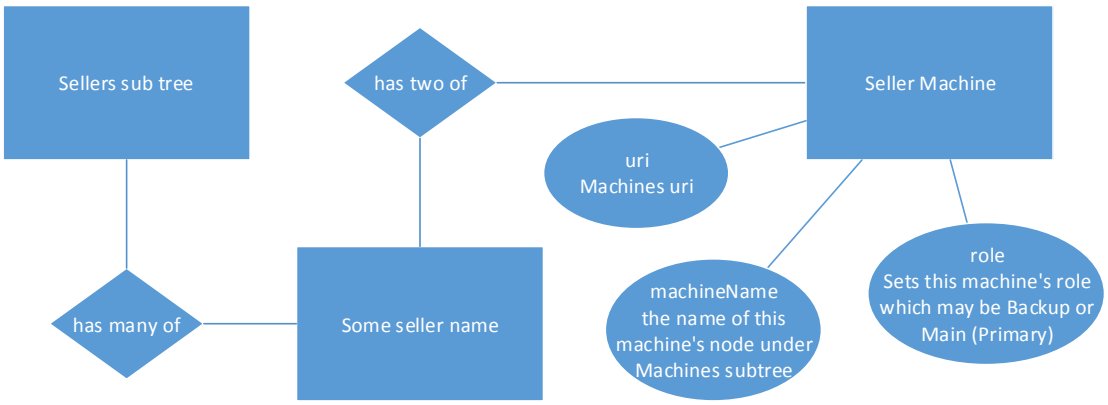
לכל מוכרן המצטרף למערכת יש לכל היותר 2 בנים המסמלים את המכונות שמחזיקות את המוכרן הזה (אחת כגיבוי ואחת ראשית). בנים אלו הם מסוג Ephemeral כך שבעת כשל הוא ימחק מהמוכרן וה-Delegate של ה-Alliance לא יראה אותו יותר ולכן ייגש ישירות לשרת חי המחזיק העתק שלו.

כל עלה בענף המכונות מחזיק זוג רשימות המתארות את המוכרנים שהוא "מחזיק" (יכול לתת שרות עבורם), רשימה אחת עבור המוכרנים שהוא משרת כרגע באופן פעיל (ה-delegate במערכת ניגש אליו עבור שאילתות) ורשימה נוספת עבור מוכרנים שהוא מחזיק כגיבוי (במידה ולמוכרן לא קיימת מכונה המוגדרת כראשית יש לגשת למכונת הגיבוי כדי לקבל שירות).

להלן תיאור המידע המוחזק ע"י עלים בענף המכונות



להלן תיאור המידע המוחזק ע"י עלים של מוכרן בענף המוכרנים



טיפול באירועים

כאשר מכונה חדשה מצטרפת או עוזבת (או נופלת...) המערכת מקבלת גירוי מה-ZooKeeper. הגירוי גורם להפעלת פונקציית ה-Callback אשר בוחנת את השינוי שגרם לגירוי מול תמונת הקבוצה הלוקאלית ובוחר להפעיל את הפעולה המתאימה לשינוי, קרי, הצטרפות מכונה או עזיבת מכונה.

בשני המקרים המערכת מעדכנת את העץ הלוקאלי של הקבוצה וקוראת לאלגוריתם הבקרה המתאים בעזרת Callback. המערכת מעבירה ל-Callback את מצב הקבוצה הנוכחי (Snapshot) וכן את המכונה שגרמה לשינוי.

בהפעלת האלגוריתם, הוא נכנס ל-Barrier של ה-Alliance ומובטח לנו שכל המכונות יגיעו אליו מהר מאחר וה-ZooKeeper דואג לקונצנזוס.

האלגוריתם מסדר מחדש את ה-Snapshot ובהתאם לסידור מבקש את המוכרנים הדרושים לו מהמכונות המחזיקות אותן כרגע (ה-Snapshot הישן). נשים לב שעד כה אף מכונה לא מחקה את המוכרנים אותה היא מחזיקה ולכן המערכת יכולה עדיין לתת שרות לבקשות נכנסות, כמו כן המערכת מבצעת שיבוץ אופטימלי כדי למזער את כמות הבקשות הנשלחות.

כאשר האלגוריתם מסיים את תהליך הבקשות הוא יוצא מה-Barrier (כלומר ממתין שכל המכונות האחרות יסיימו פאזה זו), מובטח לנו שכל המכונות יגיעו לנקודה זו מאחר והאלגוריתם שרץ הינו דטרמיניסטי ומתכנס. ביציאה מה-Barrier ה-Callback שהריץ את האלגוריתם המתאים מעביר למערכת את ה- Snapshot החדש, המערכת מעדכנת את מצב הקבוצה המקומי ובנוסף מעדכנת ב-ZooKeeper רק את השינויים שנוגעים אליה, דבר אשר חוסך מאתנו נעילות של ענפי מידע ב-ZooKeeper. שינויים אלו מתבצעים בפרק זמן קצר מאוד שכן המעבר הוא על משתנה מקומי המחזיק את פרטי הקבוצה והעדכון שנשלח ל-ZooKeeper מעדכן **רק** את הרשומות של המכונה.

מאחר והשינוי מהיר ומתבצע ביציאה מה-Barrier הוא יבוצע באופן כמעט מידי ע"י כל המכונות יחד והמעבר ממצב של נפילה (שעדיין ניתן לקבל בו שרות) למצב מעודכן הינו מידי. בתום תהליך זה כל מכונה "זורקת" את המידע שכבר אינו רלוונטי אליה (כזה שהיא אינה אמורה להחזיק) ונרשמת מחדש לקבלת אירועים.

תיאור מודולרי של התכנית

לקוח

מאפשר ביצוע שאילתות לטיסות עפ"י מועד יציאה, יעד, מוצא ופרמטר אופציונלי הבוחר את המוכרן אליה יש להפנות את הבקשה. אם הבקשה לא מפנת למכרן מסוים היא תשלח לכל המוכרנים ולכל ה Alliances הרשומים למערכת החיפוש.

הלקוח יציג את תוצאות החיפוש ממוינות עפ"י מחיר שפורט במסמך הדרישות של התרגיל.

שרת FlightSearchServer

שרת זה מהווה אגרגטור בקשות של לקוחות ומעביר אותם ל-Delegates הרשומים אצלו. ה-Delegates עונים לו ומחזירים רשימות של טיסות המתאימות לקריטריון והוא מאחד, ממיין ומחזיר את הרשימות ללקוח. כאשר Delegate נופל תפקיד ה-Delegate החדש להירשם מחדש.

אם שרת החיפוש פנה ל- Alliance בזמן שהמערכת מבצעת פעולת איזון אזי ייתכן כי המערכת תחזיר תשובה לא מלאה (יורחב בהסבר על שרת AirlineServer).

שרת AirlineServer

מודול TreeView

תפקידו של מודול זה הוא לנהל את ה-Group Membership View הלוקאלי ולקרא לפונקציות callback מתאימות כאשר יש שינוי במערכת (הצטרפות או נפילת מכונה) ולעדכן את העץ הלוקאלי ואת עץ ה-ZooKeeper באופן סלקטיבי. המודול מורכב ממספר תת מודולים בהם ה-TreeView עצמו שהינו מסד נתונים מקומי המשקף את מצב המערכת ומודול AirlineReplicationModule שמנהל את החיבור מול שרת ה ZOOKEEPER, מאזין לשינויים בענף המכונות של ה-Alliance אליו הוא שייך ומבצע פעולות Merge על ה TREEVIEW מאירועים שהוא מקבל, באופן זה ה-View הלוקאלי תואם תמיד לזה של ה-ZooKeeper.

AirlineReplication חושף ממשק אירועים (Callbacks) וממשק פונקציונלי שבעזרתו ניתן לקבל תמונת מצב לוקאלית של ה-Alliance ולהשפיע על מצב המערכת ועל כניסות בעץ ה-ZooKeeper **השייכות למכונה שרצה**, כלומר, מכונה שרצה אינה יכולה להשפיע על מבני נתונים שאינם שייכים באופן מפורש אליה. כשמתבצעת בקשת שינוי למבנה נתונים כזה השינוי יתבצע באופן לוקאלי בלבד ולא יישלח ל-ZooKeeper.

מימוש זה מאפשר להפריד באופן מוחלט את אלגוריתם ה- Load Balancing וטיפול בנפילות (שהם בעצם הינו הך מאחר וכשמטפלים בנפילות עושים זאת עם שיקולי עומס).

כאשר המערכת מתעוררת היא מבצעת אתחול לאובייקט הרפליקציה (AirlineReplicationModule). כאשר אובייקט הרפליקציה מאותחל הוא מבצע אנומרציה לענפים בעץ הרלוונטיים אליו, מוסיף את המוכרן השייך לו באם לא קיים עבורו ענף, נרשם לאירועי שינוי בעץ ומצטרף בעצמו לענף המכונות והמוכרנים של אותו Alliance אליו הוא שייך.

הצטרפות זו גוררת אירוע שינוי בעץ, דבר אשר מפעיל את פונקציית הטיפול בהצטרפות מכונה חדשה. פונקציית ההצטרפות בוחנת את השינויים בעץ הלוקאלי, מבצעת עליו Diff Merge, קוראת לפונקציית callback ומעבירה לה Snapshot של המערכת ופרטי המכונה שהצטרפה ובסיום הקריאה היא נרשמת מחדש לשינויים בענף המוכרנים בעץ.

כאשר מכונה נופלת מופעלת פונקציית העזיבה אשר בוחנת את השינויים בעץ הלוקאלי, מבצעת עליו Diff Merge, קוראת לפונקציית callback ומעבירה לה Snapshot של המערכת ופרטי המכונה שנפלה ובסיום הקריאה היא נרשמת מחדש לשינויים בענף המוכרנים בעץ.

בשני המקרים, הדרך של ה-Callback להשפיע על מצב המערכת הוא בעזרת פונקציה בשם updateMachineData אשר מקבלת פרמוטציה על ה-Snapshot אותו ה-Callback קיבלה.

הפונקציה updateMachineData בוחנת את ההבדלים בין המידע שהיא קיבלה לבין העץ הנוכחי ומבצעת עדכונים **רק היכן שיש שינוי** (מטעמי אופטימיזציה) ומעדכנת בעץ ה-ZooKeeper רק את הרשומות בהן המכונה הופיעה ושוב, **רק היכן שיש שינוי** כדי לחסוך בקריאות מיותרות. הגישה ב-ZooKeeper רק לרשומות שבבעלות המכונה חוסכת מאתנו לתחזק מנגנוני נעילה ומספקת אופטימיזציה בגישות לרשת.

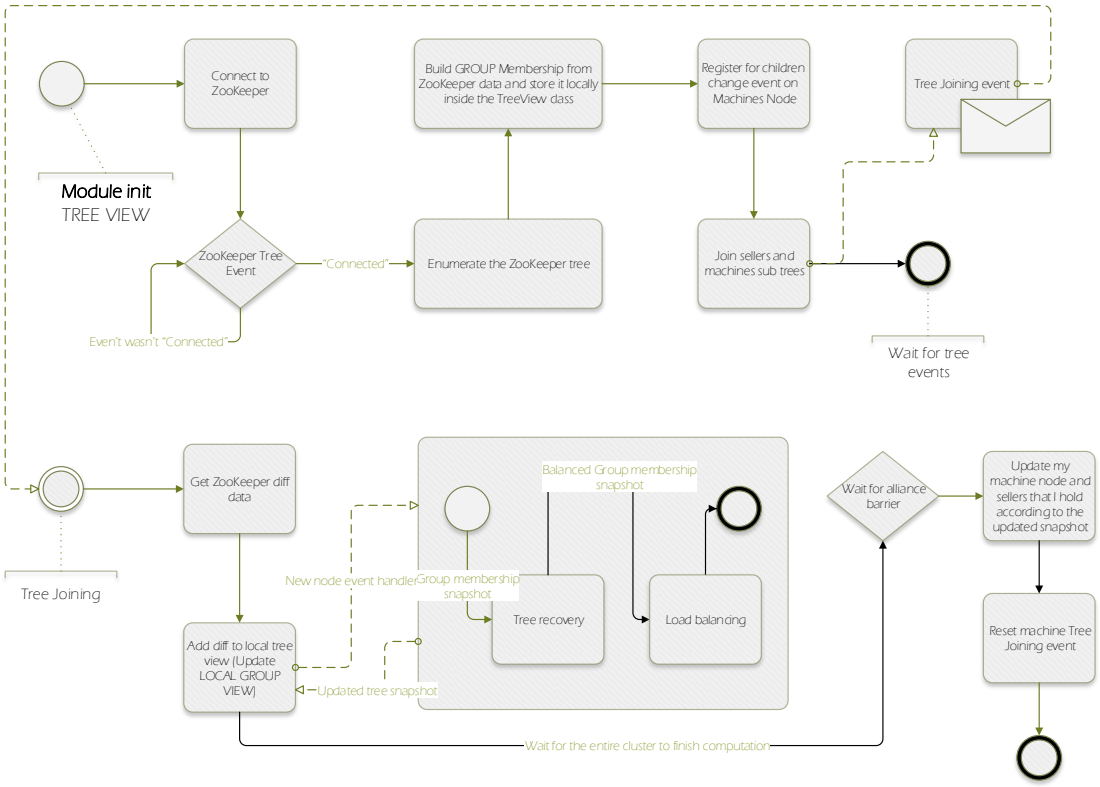
המידע שעלול להשתנות ב-ZooKeeper בעקבות הקריאה לפונקציה זו הוא:

1. הסרת עלה השייך למכונה בענף המוכרנים (כי אלגוריתם הניתוב החליט שהמכונה כבר אינה גיבוי\ראשית).
2. שינוי רשומת המכונה בענף המוכרנים (המכונה עברה בין המצבים BACKUP ו-PRIMARY).
3. צירוף המכונה כעלה חדש במוכרן במידה ולא היה קיים לפי הפרמטרים שמתאימים לנתונים ב-Snapshot החדש שקיבלה (Main/Primary).
4. שינוי רשומת המכונה בענף המכונות.

נשים לב כי ל-ZooKeeper יש 2 תפקידים עיקריים:

1. החזקת Group Membership שבעזרתו מכונות חדשות יכולות לקבל תמונת מצב של המערכת ולבנות Snapshot.
2. מערכת גירוי המספקת אירועים ומידע בהצטרפות\עזיבה של תחנות.

להלן סכמה המתארת את עליית המודול ומודל לטיפול באירוע עליה של מכונה חדשה:



מודול ראשי

מודול AirSellerRegistration

מודול זה אחראי על הרשמה של שרת airline ל- search server כ-delegate.

הוא מספק 2 פונקציות: register, unregister.

<u>Register</u>: מקבל שם של קלאסטר ל-URI של המכונה שנרשמת – אם קיימת כבר מכונה delegated שרשומה עבור הקלאסטר – הסר אותה וסגור את החיבור.

הפונקציה פותחת חיבור SOAP עבור השרת delegate עם ה-search server על מנת לבצע חיפושים כנדרש.

Unregister – מאפשרת הסרה של שרת delegate ממערכת ה-search server.

מודול ה-Cache

מודול זה אחראי על מנגנון המטמון.

החלטנו להוסיף מנגנון cache עבור שרתי ה-airline servers. את פעולות ה-cache יבצע רק שרת המוגדר כ-delegated. ה-cache מורכב מאינדוקס שאילתות המגיעות מה-search server (יצירת מחרוזת הבנויה חח"ע מהשאילתה) וערכי ההחזרה של החיפוש. בכל שאילתה המודול בודק האם היא קיימת ב-cache, אם כן הוא מחזיר את התוצאה מה-cache, אחרת מריץ את החיפוש כרגיל.

עבור הצטרפות שרת ל-Alliance – מוחקים את ה- cache כי התוצאות הישנות לא רלוונטיות (נוספו עוד מסלולים אפשריים לתוצאה).

עבור נפילה של שרת – אין שינוי מכיוון שגיבוי שלו מתוחזק ב-Cluster (קרי, Alliance).

החלטנו לממש את מנגנון ה-Cache ב-delegate מאחר ולדעתנו זו הנקודה הקריטית בה יש לנו "מספיק אך לא יותר מדי" מידע, הצטרפויות ב-Alliance אחר (הגוררות מחיקת מטמון עבור אותו Alliance) לא תשפיע על מטמון של-Alliance אחר.

אם היינו בוחרים לממש את המנגנון בשרת החיפוש אזי שרת החיפוש היה צריך לאחסן מידע בנוגע לשינויים ב-Alliances (הצטרפות) או להוסיף הודעה ב-API. מאחר ואין מספיק מידע לגבי השרתים הספציפיים בשרת החיפוש כל הצטרפות מכונה ל-Alliance כלשהו תגרור מחיקת ה-Cache הכללי של המערכת, כלומר פגיעה ב-Alliances אחרים שהבקשות שלהם במטמון.

אין משמעות ל-Cache בקונטקסט של תקשורת פנימית ב-Alliance.

אלגוריתמים

הצטרפות שרת:

1. היכנס ל-barrier.
2. משוך snapshot עדכני (שרתים + מוכרנים). (הערה: השרת שנוסף יופיע בתמונה והמוכרן שאתו הוא עלה יופיע תחת primary).
3. הסר מה-snapshot את כל המוכרנים בעלי שם זהה למוכרן שהתווסף (הסרת גרסאות ישנות).
4. אם השרת הנוכחי מכיל עותקים של מוכרנים בעלי שם זהה לזה שהתווסף – הסר אותם.
5. שמור עותק של snapshot (ייקרא snapshotOld) – על מנת לקבל תמונת מצב קונסיסטנטית המבטיחה קיום של מוכרן בכל שרת שמצוין בתמונה.
6. אם השרת שעלה הינו הראשון במערכת – הירשם כ-delegate ב-search server.
7. אם השרת שעלה הינו השני במערכת – הבא את כל ה-primaries של השרת הראשון כ-backups בשרת השני ולהפך.
8. אחרת – מיין את השרתים בסדר הפוך לפי עומס, שבץ את המוכרן שעלה כ-backup בשרת הכי פחות עמוס.
9. הרץ את אלגוריתם איזון העומסים (לאחר עליה של שרת).
10. צא מה-barrier.
11. עדכן את ה-zoo keeper (לפי ה-snapshot שנערך).

12. הסר את כל העותקים של המוכרנים שלא מוגדרים כשייכים לשרת לפי ה-ZK.

איזון העומסים לאחר עליה של שרת:

1. חשב את הממוצע הרצוי של מס' המוכרנים בכל שרת (primary ו-backup בנפרד).
2. מיין את השרתים לפי העומס של ה-primaries עליהם. עבור כל שרת לפי הסדר:
   a. כל עוד מספר השרתים כ-primary גדול מהממוצע, השרת שהצטרף יבקש מוכרן כלשהו (דטרמיניסטי – הבחירה מסתמכת על מיון בכל רגע נתון) ויעביר אותו אליו (יעדכן את המעבר ב-snapshot).
   b. אם השרת שאמור לקבל את ה-Primary הוא השרת הנוכחי – בקש אותו משרת שידוע שהוא מחזיק את המוכרן לפי ה-snapshotOld
3. מיין את השרתים לפי העומס של ה-backups עליהם. עבור כל שרת לפי הסדר:
   a. כל עוד מספר השרתים כ-backups גדול מהממוצע, השרת שהצטרף יבקש מוכרן כלשהו שאינו נמצא ב-primaries שלו (דטרמיניסטי – הבחירה מסתמכת על מיון בכל רגע נתון) ויעביר אותו אליו (יעדכן את המעבר ב-snapshot).
   b. אם השרת שאמור לקבל את ה-backup הוא השרת הנוכחי – בקש אותו משרת שידוע שהוא מחזיק את המוכרן לפי ה-snapshotOld

נפילת שרת:

1. היכנס ל-barrier.
2. אם השרת לא היה delegated עד עכשיו, אבל ה-machineName שלו הכי קטן לקסיקוגרפית – סמנו כ-delegated והירשם כ-delegate ב-search server.
3. משוך snapshot עדכני (שרתים + מוכרנים). (ללא השרת שנפל).
4. שמור עותק של snapshot (ייקרא snapshotOld) – על מנת לקבל תמונת מצב קונסיסטנטית המבטיחה קיום של מוכרן בכל שרת שמצוין בתמונה.
5. הרץ את אלגוריתם איזון העומסים (לאחר נפילה של שרת).
6. צא מה-barrier.
7. עדכן את ה-zoo keeper (לפי ה-snapshot שנערך).
8. הסר את כל העותקים של המוכרנים שלא מוגדרים כשייכים לשרת לפי ה-ZK.

איזון העומסים לאחר עליה של שרת:

קלט האלגוריתם: רשימת המוכרנים שאיבדו את השרת primary שלהם וכן רשימת המוכרנים שאיבדו את השרת backup שלהם.

1. חשב את הממוצע הרצוי של מס' המוכרנים בכל שרת (primary ו-backup בנפרד).
2. מיין את השרתים לפי הזמינות (ההפך מעומס) של ה-primaries עליהם. עבור על כל שרת לפי הסדר:
   a. כל עוד מס' המוכרנים המוגדרים כ-primaries עבור השרת קטן מהממוצע – הוסף אליו מוכרן כלשהו מ "שרתים שאיבדו את ה-Primary שלהם" (דטרמיניסטי לפי מיון לקסיקוגרפי).
   b. אם המוכרן שהוסף נמצא ברשימת ה-backups, הסר את המוכרן מרשימה זו והוסף אותו לרשימת shouldBeMoved וגם "שרתים שאיבדו את ה-backup שלהם".
   c. אם השרת שאמור לקבל את ה-Primary הוא השרת הנוכחי – בקש אותו משרת שידוע שהוא מחזיק את המוכרן לפי ה-snapshotOld .

d. אם "רשימת המוכרנים שאיבדו את השרת primary שלהם" התרוקנה – המשך הלאה.

3. מיין את השרתים לפי הזמינות (ההפך מעומס) של ה-backups עליהם. עבור על כל שרת לפי הסדר:

a. כל עוד מס' המוכרנים המוגדרים כ-backups עבור השרת קטן מהממוצע – הוסף אליו מוכרן כלשהו מ "שרתים שאיבדו את ה-backup שלהם" (דטרמיניסטי לפי מיון לקסיקוגרפי), בתנאי שאינו נמצא ב-Primaries שלא אותו שרת. (אם אין כזה – עבור לשרת הבא לפי הסדר).

b. אם השרת שאמור לקבל את ה-backup הוא השרת הנוכחי:

i. אם המוכרן שהוסף נמצא ברשימת ה-shouldBeMove , בקש אותו משרת שידוע שהוא מחזיק את המוכרן כ-backup לפי ה-snapshotOld (מכיוון שדרשנו העברה שלו משרת אחר שהחזיק אותו כ-backup).

ii. אחרת - בקש אותו משרת שידוע שהוא מחזיק את המוכרן כ-primary לפי ה-snapshotOld (מכיוון שאין לו backup עדין).

c. אם "רשימת המוכרנים שאיבדו את השרת backup שלהם" התרוקנה – סיים.

הערות:

- לכל אחד מהשרתים קלט זהה (באמצעות ה-ZK). כל אחד מהשרתים מריץ עצמאית אלגוריתם דטרמיניסטי שמבטיח שבהנתן קלט זהה נקבל פלט זהה ובכך שומרים על קונצנזוס.
- הדטרמיניזם נובע מכך שכל בחירה נקבעת באופן יחיד על סמך מיון.
- על מנת להבטיח שמנגנון העברת ה-data של המוכרנים לא יכשל בגלל בעיות סנכרון, כל שרת שמבקש מוכרן משרת אחר "מגלה" איזה שרת מחזיק איזה מוכרן על סמך ה-snapshotOld .
- כל שרת ידאג לא להסיר data של מוכרנים ממבנה הנתונים שלו עד שכל שאר השרתים התייצבו ובכך מבטיח זמינות של data עבור כל בקשה משרת שאחר.
- ברגע ששרת סיים את אלגוריתם ההתאוששות הוא יבקש לצאת מה-barrier. באמצעות ה-barrier כל השרתים ידעו שכולם סיימו "להעביר הודעות" מאחד לשני ולכן אפשר לעדכן את ה-ZK לפי פלט האלגוריתם וכן להסיר את כל ה-data של המוכרנים שאינם נחוצים יותר לפי האלגוריתם.
- ה-load balancing מובטח עבור primaries ו-backups בנפרד וזאת מכיוון שלכל אחד תפקיד אחר. תכונה זו מובעת מאופן פעולת האלגוריתם. כלומר, חישוב ממוצע ובחירות דטרמיניסטיות של שרתי למוכרנים. כמו כן, בנפילות נדרשות העברות של מוכרנים בין שרתים כדי להבטיח איזון עומסים אופטימלי.
- ה-delegate ב-search server נבחר על אותו עיקרון של leader שנלמד בשיעור (מינימום לקסיקוגרפי עבור שם השרת).
- אנו מבטיחים סילוק גרסאות ישנות בכך שהאלגוריתמים מוחקים מייד כל "גרסה ישנה" של מוכרן שזה עתה עלה.

ניתוח ביצועים

חיפוש

הבקשה שמגיעה ל-FlightSearchServer נשלחת לכל ה-Delegates במערכת (נסמנם d). אם הלקוח פירט מוכרנים כל Delegate יבחן את רשימת המוכרנים שלו וישלח ל-Primary של המוכרן והתשובות יוחזרו באותו מסלול ללקוח.

אם הלקוח לא פירט מוכרנים ה-Delegate ישלח את הבקשה לכל השרתים פעמיים, פעם שאילתה לפי מוצא ותאריך ופעם נוספת לפי תאריך ויעד וזאת כדי להרכיב טיסות Connection.

התלבטנו האם ה-Delegate יתשאל בעצמו כל אחד ואחד מהשרתים עבור טיסות לפי מוצא ויעד ויבצע את ההתאמה בין טיסות (בכדי לבנות Connection) לבין האפשרות שה-Delegate ישלח הנחייה לכל השרתים הרלוונטיים ב-Alliance כך שכל אחד מהם מחפש טיסה לפי יעד בשרת אחר ויבצע את ההתאמה בעצמו.

יתרון האפשרות הראשונה הוא חיסכון בתקשורת, סיבוכיות ליניארית לעומת סיבוכיות ריבועית במחיר העמסת ה-Delegate. אנו בחרנו באפשרות הראשונה. ובסה"כ הסיבוכיות עבור כל Alliance ליניארית במספר השרתים ובסה"כ עבור כלל המערכת $O(n')$ כאשר n' הוא סך כל השרתים במערכת.

שינויים עקב עליה\נפילה

לא נפריד את הדיון לעליה וירידה מאחר ושני האלגוריתמים מבצעים פעולה דומה (חישוב מקומי ובקשת העברות).

בעת שינוי במערכת נכנסים ל-Barrier, דבר הקורה מהר מאוד מאחר וכל המכונות רואות את השינוי (מנגנון ה-ZooKeeper מבטיח זאת), נסמן זמן זה ב- ε.

עבור כל מוכרן האלגוריתם יבצע לכל היותר העברה אחת משרת לשרת, כלומר, אין העברת n העתקים של מוכרן בין כל המכונות ולכן תהליך זה חסום במספר המוכרנים. האלגוריתם הינו דטרמיניסטי והעברת כל המוכרים בסה"כ תבוזר בין כל המכונות באופן זר ולכן החסם העליון למכונה הוא $O\left(\frac{n}{m}\right)$ כאשר n הוא מספר המוכרנים ו-m מספר המכונות ב-Alliance. כמובן שבמימוש האלגוריתם נקטנו באופטימיזציה אגרסיבית כך שבד"כ הסיבוכיות היא נמוכה יותר.

לאחר הרצת האלגוריתם, המכונה מעדכנת את תתי עצי המוכרנים של הרשומות בהן היא הופיעה או מופיעה כעת וגם את הכניסה המתאימה לה בתת עץ המכונות, ובסה"כ מהחסם על סיבוכיות האלגוריתם נקבל - $O\left(1+\frac{n}{m}\right)$

ובסה"כ, האלגוריתם חסום מלמעלה - $O\left(\frac{n}{m}\right)$.

ממשקים – API

ממשק שרת חיפוש

בפני לקוחות

ממשק מסוג **REST** החושף את הפונקציונליות הבאה:

- **/Services/FlightsSearch/flight?src={src}&dst={dst}&date={date}&servers={servers}**
  - פונקציונליות לקוח
  - סוג GET
  - *ערך חזרה:* מחזירה רשימה של תוצאות
  - *תיאור:* הפונקציה שולחת שאילתה לשרת החיפוש. הפונקציה תחזיר רשימה של טיסות (כולל טיסות עם קונקשן) ממקור src ליעד dst בתאריך date מהשרת servers.

בפני שרתי מוכרנים ו-Alliance delegates

ממשק מסוג **REST** החושף את הפונקציונליות הבאה:

- **Services/FlightsSearchReg/Register/{clustername}**
  - Cluster uri – Uri
  - סוג PUT

  שרת delegate של Alliance מסוים או מוכרן כלשהו המעוניין לקבל בקשות חיפוש מהשרת מפעיל בקשה זו.
- **Services/FlightsSearchReg/Unregister**
  - אין קלט
  - *תיאור:* הפונקציה תשתמש בכתובת המקור ממנה הבקשה הגיעה כדי להסיר את ה Alliance מרשימת השרתים שאליהם שולחים חיפושים.

ממשק שרת Airline server

בפני שרתי חיפוש ומוכרנים המתפקדים כ-Delegates של Alliance:

ממשק מסוג **SOAP** החושף את הפונקציונליות הבאה:

- **/Services/SellerService/getTrips**
  - Src – מקור הטיסה (String)
  - Dst – יעד הטיסה (String)
  - Date – תאריך הטיסה (DateTime)
  - Sellers – אופציונלי, אם הלקוח פירט רשימת מוכרנים מסוימת (List<String>)
  - *ערך חזרה:* List<Trip> כאשר Trip הינו אובייקט המכיל את כל המידע שנדרש להצגה אצל הלקוח.
  - *תיאור:* השרת המתפקד כdelegate חושף שירות זה המאפשר ביצוע שאילתה על ה-Alliance. אם הוגדרו מוכרנים וה-delegate אינו מכיר אותם הוא יתעלם מהבקשה ויחזיר רשימה ריקה.

ממשק תחזוקה פנימי בין שרתי מוכרנים

ממשק מסוג **SOAP** החושף את הפונקציונליות הבאה:

- **/Services/IntraClusterService/sendPrimarySeller**
  - sellerName – שם המוכרן (String)
  - *ערך חזרה:* מסד נתונים המייצג את כל המידע אותו המוכרן מחזיק.
  - *תיאור:* פונקציה זו מבקשת מהמכונה המריצה את השרות את מוכרן ראשי אותו היא מחזיקה (יתכן שבעת נפילות מכונה מחזיקה מספר מוכרנים כראשיים – כלומר פעילים), פונקציה זו משמשת מוכרנים אחרים בעת ביצוע תהליך Load

Balancing בעת נפילת או הצטרפות מכונה חדשה. ערך החזרה עלול להיות גדול מאוד שכן הוא מייצג את מסד הנתונים כולו.

- **/Services/IntraClusterService/`sendBackupSeller`**
- sellerName – שם המוכרן (String)
- *ערך חזרה:* מסד נתונים המייצג את כל המידע אותו המוכרן מחזיק.
- *תיאור:* פונקציה זו מבקשת מהמכונה המריצה את השרות את מוכרן גיבוי אותו היא מחזיקה (יתכן שבעת נפילות מכונה מחזיקה מספר מוכרנים כגיבוי), פונקציה זו משמשת מוכרנים אחרים בעת ביצוע תהליך Load Balancing בעת נפילת או הצטרפות מכונה חדשה. ערך החזרה עלול להיות גדול מאוד שכן הוא מייצג את מסד הנתונים כולו.
- **/Services/IntraClusterService/`getRelevantFlightsBySrc`**
- Src – נקודת היציאה של הטיסה (String).
- Date – תאריך היציאה של הטיסה (DateTime)
- *ערך חזרה:* רשימת טיסות <List<Flights שיוצאות מ-src בתאריך Date.
- *תיאור:* פונקציה זו נקראת ע"י ה-Delegate של ה-Alliance במטרה לנסות להרכיב טיסות היוצאות בתאריך ה-Date (הרכבת קונקשן).
- **/Services/IntraClusterService/`getRelevantFlightsByDst`**
- Dst – יעד הטיסה (String)
- Date – תאריך היציאה של הטיסה (DateTime)
- *ערך חזרה:* רשימת טיסות <List<Flights המגיעות ל-Dst ויוצאות בתאריך Date.
- תיאור: פונקציה זו נקראת ע"י ה-Delegate של ה-Alliance במטרה לנסות להרכיב טיסות היוצאות בתאריך ה-Date (הרכבת קונקשן).

חלק יבש

שינוי פונקציונלי:

נציע מימוש לפעולות reserve ו-cancel אך לפני כן נניח כי עבור טיסות עם קונקשן הלקוח ישלח בשאילתה את מספרי הטיסות (מקור-קונקשן, קונקשן-יעד).

אין שינוי בפרדיגמת התקשורת בין הלקוח לבין שרת החיפוש ובין שרת החיפוש לבין הdelegate של ה-cluster (הלקוח ישלח את הבקשה ושאר הישויות בדרך ישמשו כ"נתב מסובך").

אם מדובר בהזמנה המערבת 2 מוכרנים שונים נבצע את הפעולות הבאות:

מאחר ושרת הdelegate הוא נקודת הכניסה והיחידה היציאה ל-cluster ומאחר ושרתים ב-cluster אינם משפיעים זה על זה כאשר מקבלים בקשת cancel/reserve מספיק לנעול את זוג המוכרנים לבקשות נוספות מסוג זה. הנעילה תתבצע ע"י הגדרת watch בכל שרת על מסלול אפשרי למנעול עבור מוכרן primary אותו הוא מחזיק (znode מסוג persistent). מוכרן נעול יכול לענות לשאילתות Query אך אינו עונה לשאילתות cancel/reserve.

מוכרן הטיסה הראשונה ינסה לבצע הזמנה במוכרן הטיסה השנייה, ובמידה ויצליח יבצע הזמנה על עצמו, יחזיר תשובה ל-delegate וימחק את המנעול שהוגדר לזוג השרתים.

אם מדובר בהזמנה המערכת מוכרן יחיד ננעל אותו גם כן וזאת בכדי שלא יהיה ניתן לבצע הזמנות\ביטולים כאשר המוכרן שמנסים לעבוד אתו הוא קונקשן של בקשה אחרת. אם הצלחנו נמחק את המנעול ונשלח חיווי.

בנפילה המכונה שמחזיקה Backup של מוכרן שהיה באמצע טרנסאקציה תמחק את המנעול שלה.

שינוי במנגנון הרפליקציה:

מעתה והלאה נתייחס להזמנה\ביטול כאותה פעולה ונכנה אותה טרנסאקציה.

נציע 2 פתרונות -

<u>גישה 1:</u>

כאשר ה-Delegate מקבל בקשת טרנסאקציה הוא ישלח את הבקשה למכונה הראשית וגם למכונת הגיבוי. באופן זה כל שינוי יתבצע על כל ההעתקים בו-זמנית.

היתרון בגישה זו הוא שהתאוששות מנפילה הינה מהירה ביותר אך החיסרון הוא שהמכונות "חמות" ויש פי-2 תקשורת.

<u>גישה 2:</u>

כאשר טרנסאקציה מצליחה המוכרנים המעורבים ישמרו בעץ ה-ZooKeeper תחת המוכרן כמות מסוימת של טרנסאקציות אחרונות (ניתן לתת סדר על בנים עפ"י ה-API של ZooKeeper) וכשנגיע לכמות המקסימלי המוכרן יזום עדכון בגיבויים שלו וינקה את היסטורית הטרנסאקציות.

השינוי באלגוריתם השכפול יבא לידי ביטוי בכך שבשלב השכפול וההעברות כל מכונה שמחזיקה מוכרן גיבוי תעדכן את המידע שהוא מחזיק בעזרת המידע בעץ ובתום שלב ההעברות (ביציאה מה-Barrier) המכונה שמוגדרת כראשית עבור המוכרן תמחק את הטרנסאקציות מהעץ.

היתרון בגישה זו שהוא חוסך בתקשורת ובפעילות המכונות אך החיסרון הוא שהתאוששות תארך יותר זמן ושאנו מעמיסים את ה-ZooKeeper בבנים.