Distributed Systems 236351 Home Assignment 2

Decentralized uber

Shai Porath 312434863 shai.porath@campus.technion.ac.il

Hagar Sheffer 205814627 hagar.s@campus.tecnion.ac.il

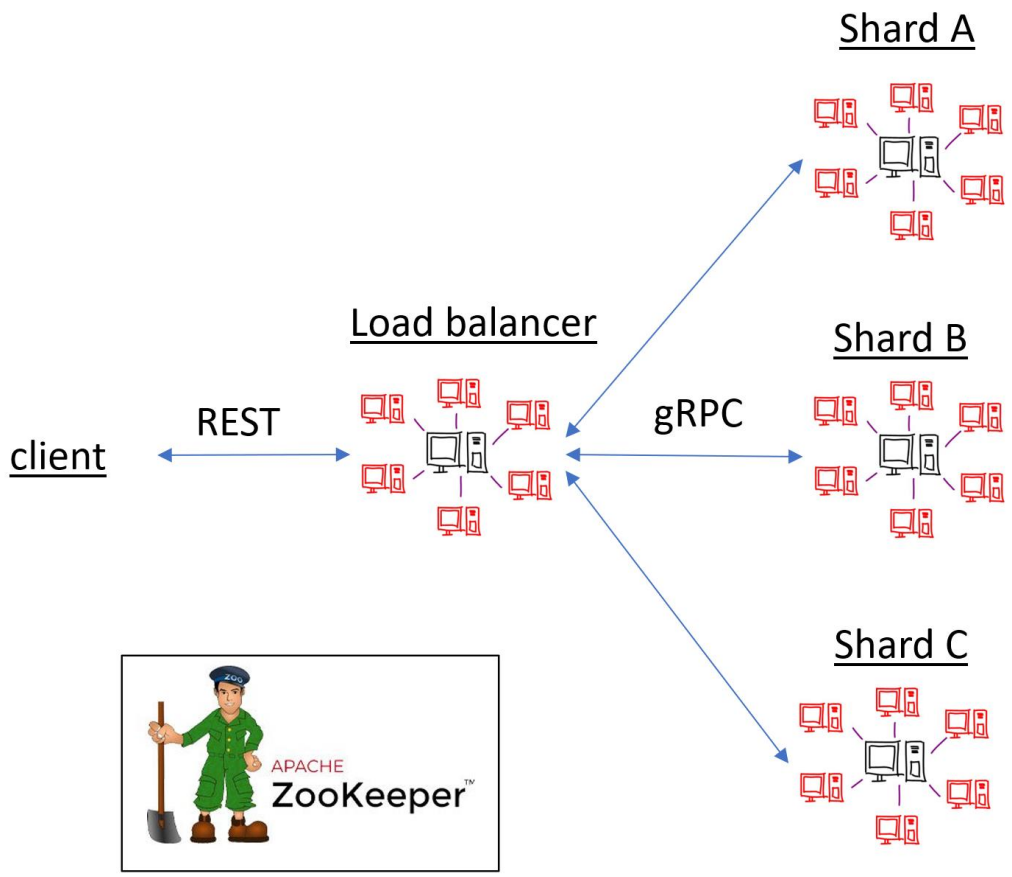
לינק לגיטהאב: https://github.com/hagar21/DistributedSystems2.git

הbranch הסופי הוא newMaster

המערכת שבנינו תומכת בשתי פונקציות:

1. Post ride – משתמש יכול להירשם כנהג במערכת עבור נסיעה בין שתי ערים בתאריך מסויים.
2. Post path planning request – משתמש יכול לפרסם מסלול העובר ברשימת ערים בתאריך מסויים. אם קיימת קבוצת נסיעות המספקת את המסלול, ישמר עבורו מקום בכל אחת מהנסיעות ונחזיר את פרטיהן. אחרת נחזיר שאין אפשרות לספק את המסלול.

מבנה הרשת



מאזן עומסים (Load Balancer, LB)

שירות זה אחראי לקבל בקשות ממשתמשים באמצעות REST ולהעביר כל בקשה לשרת האיזורי המתאים, תוך התחשבות בשני פרמטרים:

1. השרת שייך לאיזור האחראי על העיר ממנה יוצאת הנסיעה המבוקשת.
2. מבין שרתי האיזור המתאים, הבקשה תישלח לשרת הכי פחות עמוס.

בנוסף, כאשר שרתי איזור שונים רוצים לדבר אחד עם השני, הבקשה תעבור דרך מאזן העומסים אשר יעביר את הבקשה לשרת הפנוי ביותר השייך לאיזור המבוקש.

איזון העומסים יתבצע בשיטת Round Robin. יתרונות בשיטה זו:

- השיטה פשוטה למימוש
- יכולות החישוב של השרתים השונים זהה
- עלות החישוב בין בקשות path planning דומה וכן ובין בקשות post ride ואנו מניחים פיזור אחיד של הבקשות.
- תקורה נמוכה – כיוון שמאזן העומסים מקבל את כל הבקשות מהמשתמשים ומעביר אותם לשרתים האיזוריים הוא מהווה צוואר-בקבוק במערכת. לכן העדפנו שהחישובים שהוא מבצע יהיו קצרים ככל הניתן.

באופן אמפירי אנחנו שואפות ליצור שיוויוניות בין הבקשות שכל שרת מקבל כי למשל בקשות Path Planning דורשות חישוב שונה וכבד בהשוואה לבקשות Post Ride.

מימוש מאזן עומסים תורם לפיזור העומסים, לשקיפות פרטי המערכת מול המשתמש וכן מהווה נקודת כניסה יחידה למערכת בה ניתן להציב חומת אש, ובכך משפר את הבטיחות. בנוסף, ע"י שימוש ב-zookeeper מאזן העומסים מעודכן בכל רגע ברשימת השרתים החיים ולא יפנה בקשות לשרת שנפל. בכך הוא תורם להתאוששות מהירה של המערכת מנפילות.

Load Balancer Fault Tolerance

לא היה לנו זמן לממש את הלוגיקה הזו, אבל היינו מממשות זאת באופן הבא:

כדי להבטיח עמידות בפני כשלונות, לload balancer אמורים להיות גיבויים המוכנים להחליף אותו במקרה שיפול באופן שקוף למשתמש.

היינו מממשות את הרפליקציות בשיטת primary-backup replication כאשר הbackup הוא ב-hot standby – כלומר ה-LB החלופי יריץ שירות gRPC ו-REST ורק המנהיג בCluster זה יקבל בפועל בקשות ממשתמשים ומשרתי איזור. כדי לשמר בצורה טובה את התקשורת בין הShards (השרתים האיזוריים) לLB, הShards יאזינו לחילוף המנהיג בCluster השייך למאזן העומסים וברגע שהוא מתחלף יעבור לשלוח הודעות למנהיג החדש. כדי להסתיר את ההחלפה של שרתי הLB מהמשתמש היינו משתמשות בשרת DNS אשר יפנה כל בקשה חיצונית למערכת למאזן העומסים המכהן.

נניח שקיימים N nodes של LB אז נוכל להתמודד עם נפילה של N-1 שרתים.

צביר איזורי (Shard cluster)

כל איזור מחזיק תחתיו מספר ערים כך שהוא אחראי לבצע בקשות היוצאות מהן. הרפליקציות בצביר האיזורי מבוססות total order – לכל צביר יש מנהיג יחיד, וכל שינוי בstate של הצביר חייב לעבור דרכו כדי שיקבע את הסדר שבו הגיעו הבקשות. כלומר, כל שרת בצביר יכול לטפל בבקשה ולפני שהוא שומר את התוצאה הוא מעביר אותה למנהיג כך שיפיץ אותה לכל חברי הצביר. נרחיב בהמשך על האופן שבו מתרחש הcommit ונראה שהוא שומר על מצב תקין בכל רגע גם במקרה של נפילות וכשלנות.

1. פרסום שירות – הנסיעות המפורסמות יאוחסנו בבסיס נתונים מקומי על שרתי האיזור אשר אחראים על עיר המקור של כל נסיעה. כלומר כל נסיעה שמורה במערכת בכל הרפליקציות ששייכות לעיר המוצא של הנסיעה, כך שיש גיבוי למידע זה.
2. צריכת שירות – באופן דומה, בקשה לתכנון מסלול נסיעה תטופל ע"י שרתי האיזור אשר אחראים על עיר המקור.
   a. עבור כל שתי תחנות סמוכות במסלול המבוקש השרת ראשית יחפש נסיעה מתאימה בבסיס הנתונים המקומי שלו ואם לא ימצא יבקש לחפש בבסיס הנתונים של כל אחד מהאיזורים האחרים.
   b. כדי לשמור על עקביות, ברגע ששרת מוצא נסיעה מתאימה לחלק מהמסלול הוא משריין אותה עבור הנוסע, כלומר מוסיף את הנוסע לרשימת הנוסעים השמורה בנסיעה ומעלה את מונה המקומות התפוסים בנסיעה.
   c. בכל שלב במסלול, אם לא נמצאה נסיעה מתאימה בבסיס הנתונים המקומי וכן בבסיסים המרוחקים, השרת עובר על רשימת הנסיעות ששוריינו ומבטל את השינויים. אם הנסיעה לא הגיעה מבסיס הנתונים המקומי, השרת מבקש מהאיזור שממנו הגיעה הנסיעה לבטל את השינויים.

   גם במקרה זה כל הבקשות שמורות בכל הרפליקציות ששייכות לעיר המוצא של הבקשה.

Fault tolerance – נניח שבכל צביר חברים n שרתים. כיוון שכל חברי הצביר הם רפליקציות, אנו יכולים להתמודד עם נפילה של n-1 שרתים מכל צביר.

Linearizability – ע"י הבטחת total order באמצעות השימוש במנהיג אנו מבטיחים linearizability.

Commit

לפני שמירת כל שינוי בבסיס הנתונים המקומי מתבצע commit בצביר.

- כל שרת שרוצה לשמור מידע, שאינו המנהיג, מעביר את המידע שהוא מעוניין לשמור למנהיג.
- לפני ביצוע commit המנהיג יוצר persistent znode המכיל את הנסיעה\בקשה להצטרף לנסיעה שאותה הוא מעוניין לשמור. כך, אם המנהיג יפול באמצע ביצוע הcommit, לא ישארו שרתים לא עקביים במצב שלהם.
- המנהיג שולח בקשה לכל חברי הצביר החיים לשמור את הנסיעה.
- המנהיג סופר כמה הודעות אישור הוא קיבל. אם קיבל הודעות אישור מכולם הוא שומר את המידע גם אצלו ומסיים את התהליך. אחרת, הוא שולח הודעות rollback לכל חברי הצביר עם בקשה לבטל את העדכון שביצעו.

- נציין שבמערכת שלנו יש שני סוגים של עדכונים: יצירת רשומה חדשה ועדכון רשומה קיימת. הסוג הנדרש מצויין בתוך הודעת הcommit. כדי לבטל יצירת רשומה חדשה יש למחוק אותה, ואילו כדי לבטל עדכון יש לשנות את המידע ברשומה בהתאם למה שהיה לפני העדכון.
- לאחר ביצוע הcommit המנהיג מוחק את הznode שיצר כדי לגבות את התהליך.
- מנהיג חדש שמתעורר ראשית בודק בעזרת הזוקיפר אם קיים commit שנקטע בעקבות נפילה של המנהיג. בעזרת התיקיה CommitBackup בזוקיפר נשמע המידע אודות הcommit שנכשל. אם קיים כזה, הוא מבצע בעצמו את תהליך הcommit, כך שאם המנהיג נפל באמצע ביצוע commit לא ישארו שרתים שלא קיבלו את הודעת הcommit וקיימת תמיכה בfault tolerance.
בנוסף נציין שקיים טיפול במקרה ששרת יקבל הודעת commit על אותו אובייקט פעמיים (למקרה שאותו שרת כן קיבל את ההודעה של המנהיג לפני נפילה).
- באופן דומה, אם המנהיג נפל באמצע הrollback המנהיג החדש ינסה לבצע commit ואם לא יצליח יבצע את הrollback באופן מלא.
- כדי להבטיח total order אנו מאפשרות רק ביצוע commit אחד בכל רגע ומשיגות זאת ע"י שימוש במנעול. כך מובטח שכל חברי הצביר מבצעים עדכונים בבסיס הנתונים שלהם בדיוק באותו סדר. כלומר כך אנו מבטיחות **לינארזביליות**.
- כיוון שאנו מחזירות תשובה למשתמש רק לאחר שכל חברי הצביר אישרו את שמירת הבקשה אנו מבטיחות **אטומיות** – או שבקשה מובטחת או שהיא נדחית על ידי כל חברי הצביר.

Zookeeper

אנו משתמשים בzookeeper כדי לממש failure detector, group membership, atomic broadcast

- כששרת מצטרף לאיזור מסויים הוא נרשם תחת ELECTION_NODE וכן תחת NODES_LIVE תחת האיזור המתאים כEphemeral Znode ושמו מורכב מה-IP שלו והפורט שהוא מאזין לו. באופן זה לא נדרש לגשת למידע השמור בצומת עצמו, זאת כדי לצמצם בתקשורת עם ה-zookeeper.
- לאחר מכן, אנחנו יוצרים listeners עבור השרת לתיקיית האזור שלו הנמצאת ב ELECTION_NODE, NODES_LIVE. בכל שינוי בתוכן התיקיות האלה מתבצע עדכון של המידע המקומי של השרת בנוגע למנהיג הצביר שלו ולחברי הצביר החיים, בהתאמה.
- atomic broadcast – כדי לקבל את התכונות של atomic broadcast בעת ביצוע commit, אנו משתמשות בpersistent znode ייעודי. אם המנהיג התחיל את הcommit ויצר את הznode בסופו של דבר כל חברי הצביר יקבלו את ההודעה (או יבצעו rollback). אם המנהיג נפל לפני שיצר את הznode אף שרת לא יקבל בקשה לcommit.

Fault tolerance –

נניח שקיימים N nodes של zookeeper כאשר N אי זוגי.

zookeeper דורש שיהיה קיים רוב כדי לתפקד בצורה טובה ולכן נוכל להתמודד עם $\left\lceil \frac{N}{2} \right\rceil - 1$ כשלונות.

ההירכיה בzookeeper: