Blockchain with Adaptive Batching

מגישים: קאסם סייאח
אשרף יאסין

המטרה של הפרויקט:

לממש מערכת ניהול של permissioned Blockchain להעברת Transactions בין שרתי הבנק הפדרלי של ארצות הברית, בצורה יעילה ובהתחשבות בתעבורת ה-Transactions כך שיצירת הבלוקים צריכה להיות אדפטיבית ולהתחשב במצב המערכת.

הנחות של הפרויקט:

1. ה blockchain נשמר על ידי מספר קבוע של שרתים אשר מכירים אחד את השני ויש אמון בין שרתים אלו.
2. לכל לקוח קיים קשר יעודי יחיד עם שרת - כלומר, כל לקוח יודע את כתובת השרת מראש, כך שאין צורך לשמור על מנגנון מתן שמות.
3. לכל עסקה יש מבנה יחיד (שולח,מקבל,סכום עסקה) שנפרט עליהם בהמשך.

Client:

המשתמשים של המערכת הם אלה שמבקשים לבצע טרנזקציות בין משתמש אחד לשני, הם שולחים את הטרנזקציה הרצויה אל אחד שרתי המערכת כדי שיטפל בה.

Server:

שרת במערכת הוא אחראי על עיבוד הטרנזקציות המתקבלות מ-clients, אשר מנסה להכניס מספר טרנזקציות בהתאם לעומס הרשת ושולח אותם ב block לשאר השרתים, וכל שרת הוא בעצמו client של zookeeper וזה כדי להשתמש בשירותי zk שמאפשרים מימוש atomicBroadcast להבטחת הסכמה על אותו סדר של בלוקים ב-BlockChain.

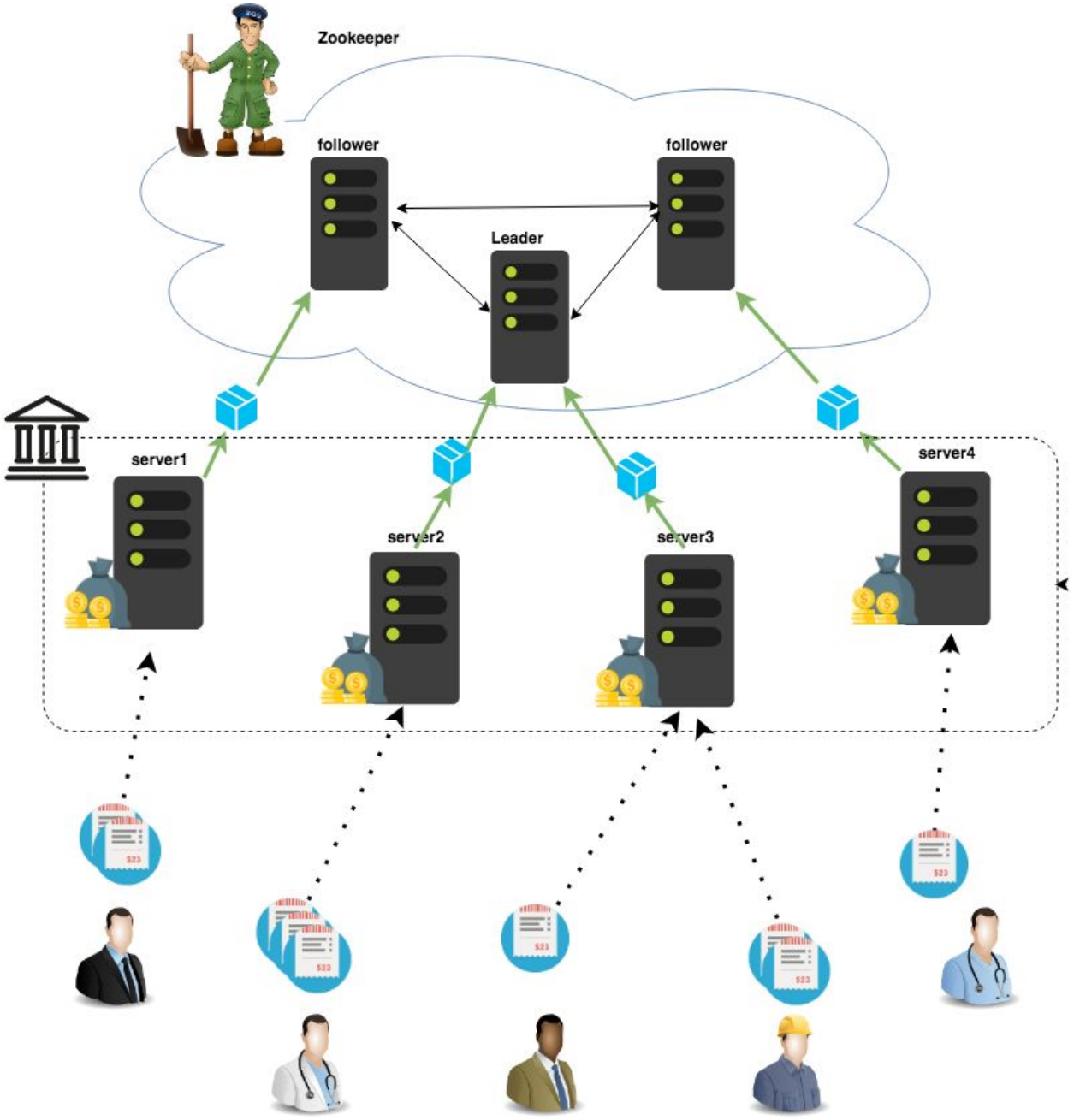
Zookeeper:

השתמשנו בשירות של zookeeper כדי לנהל את השרתים המבוזרים שלנו ולסנכרן ביניהם, המטרה העיקרית היא לקבוע סדר הבלוקים המוכנסים אל ה-replicated BlockChain.

Cleaner:

זהו Server יחיד שאחראי על ניקוי תיקיית BLOCKS/ ב-zk, הוא מוחק בלוקים שהם כבר הגיעו לכל שרתי המערכת.

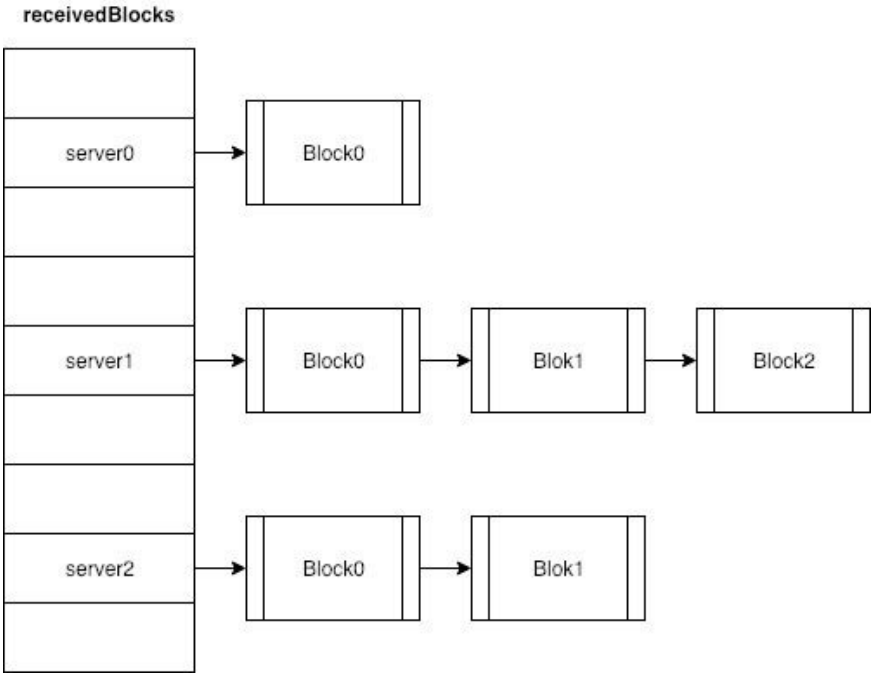
הסכימה של המערכת:



מבנה ה-server:

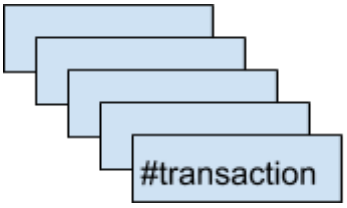
Received Blocks:

hashMap שמכיל את הבלוקים שהתקבלו מהשרתים ועדיין לא נכנסו ל-blockChain, המפתח הוא ה-ip:port של השרת והערך הוא רשימת הבלוקים שהתקבלו משרת זה.



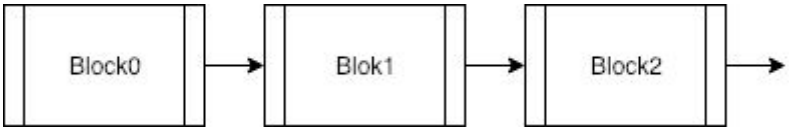
Transactions List:

רשימת טרנזקציות שמגיעות לשרת מהלקוחות.



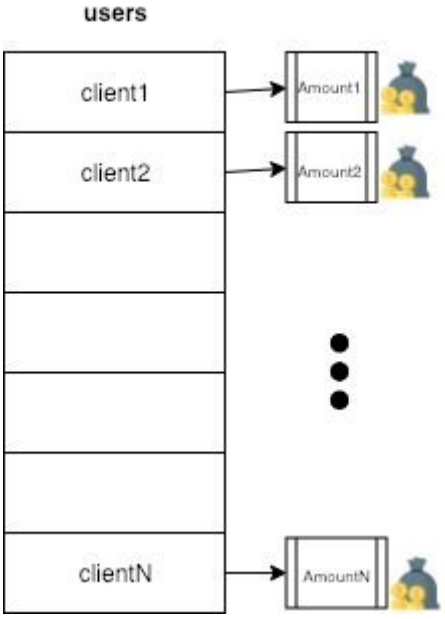
Blocks Chain:

רשימת שרשרת הבלוקים שהוסכם עליהם.



Users:

hashMap של משתמשי המערכת, המפתח הוא המזהה של המשתמש וה-value הוא ה-balance שלו.

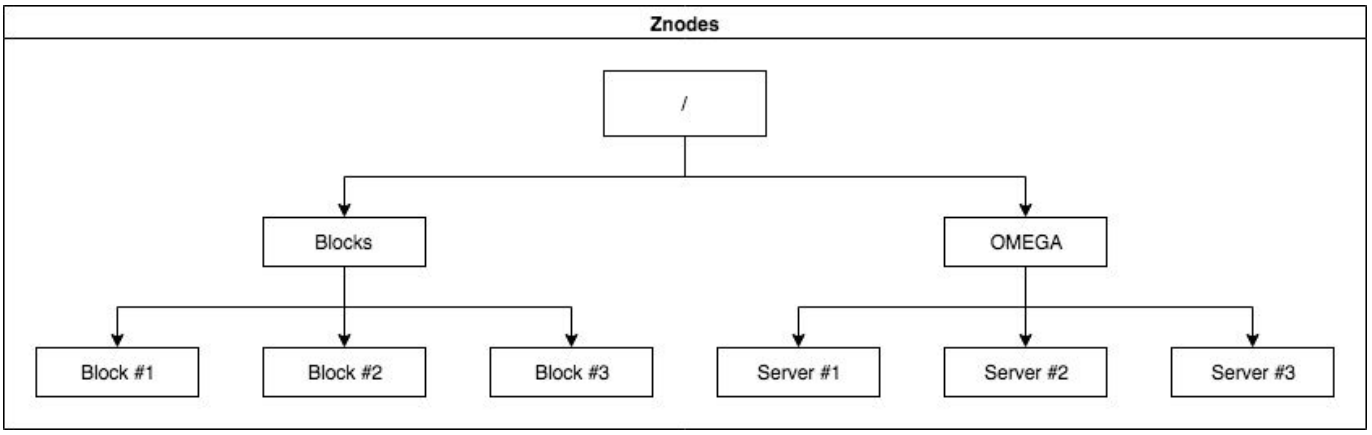


כל משתמש מאותחל עם balance התחלתי, ה-Validity של טרנזקציה נבדקת יחסית ל-balance של כל משתמש, טרנזקציה היא חוקית אם ה-balance של הצד המשלם שהוא לפחות כסכום התשלום של הטרנזקציה, ה-balance תמיד מעודכן לפי כל הטרנזקציות שהתקבלו עד כה בבלוקים ב-Block Chain.

בלוק הוא VALID אם אפשר לבצע כל הטרנזקציות שלו באותו סדר שהתקבלו בו, כלומר הוא VALID אם אפשר לבצע את הטרנזקציות אחת אחרי השניה באופן חוקי.

מבנה ה-zookeeper:

במערכת הקבצים שמתחזק zk יש שתי תיקיות ראשיות:



1. **/OMEGA** : מכילה node עבור כל server במערכת עם הזהה שלו ip:port.
משמש למימוש failure detector ולקביעת שרת מנהיג שמשמש כ- Cleaner.
כל ה-nodes בתוך התקייה הם **EPHEMERAL_SEQUENTIAL.**

2. **/BLOCKS** : מכילה node עבור כל בלוק שנשלח לכל השרתים ומחכה לתורו להיכנס ל-BlockChain, המידע בתוכו הוא המזהה של הבלוק ip:port:blockId , ה-ip:port הוא הכתובת של השרת ששלח את הבלוק ו-blockId הוא המספר הסידורי של הבלוק לפי הבלקים שנשלחו רק משרת זה.
משמש למימוש consensus על סדר הבלוקים ב-BlockChain.
כל ה-nodes בתוך התקייה הם **PERSISTENT_SEQUENTIAL** .

השתמשנו ב **zookeeper ensemble** עם **3 שרתים,** המשכפלים את המידע ב zookeeper, כאשר הם משוכפלים על שלוש containers לתמוך בנפילות תוך שמירה על מצב המערכת, עם חשיבות לשמור על מספר אי זוגי של שרתים ב zookeeper ע״י ה **swarm manager** שנדבר עליו בהמשך.

פעולות Grpc של שחקני המערכת:

- **פעולות ה-Client:**

1. **שליחת טרנזקציה:** הלקוח בונה הטרנזקציה ושולח אותה ב Grpc לשרת המקושר אליו, הפעולה בשם Send Transaction שמקבלת טרנזקציה ומעבירה אותה לשרת.

- **פעולות השרת:**

1. **קבלת טרנזקציות מהלקוחות:** ברגע שהשרת מקבל טרנזקציה מהלקוח הוא מוסיף אותה לרשימת ה Transactions List אצלו.
2. **שליחת בלוק:** בכל פעם שהשרת בונה בלוק הוא שולח אותו לשאר השרתים במערכת.
3. **קבלת בלוק:** כששרת מקבל בלוק חדש משרת אחר הוא שומר אותו ב Received Blocks אצלו.
4. **בקשת ה-sequence משרת:** המנהיג Cleaner מבקש משרתי המערכת את הsequence המקסימלי של הבלוקים ב Blocks chain אצלם.
5. **שליחת ה-sequence של בלוק:** כשה- Cleaner מבקש את ה-sequence השרת מחזיר לו כתשובה את הערך המקסימלי השמור אצלו.

שתי פעולות ה-server העיקריות:

- **Block Broadcast:**

כששרת מעוניין להכניס בלוק לשרשרת, הוא בהתחלה שולח אותו לכל השרתים במערכת שמכניסים אותו אל receivedBlocks אל רשימת הבלוקים המתאימה לו ואז הוא מייצר zkNode עבורו בתוך תיקיית BLOCKS, שמכיל רק את המזהה של הבלוק שהוא ip:port:blockId.

- **Block Receive:**

השרת מקבל watch מה-zk על כך שיש שינוי בתיקיית BLOCKS, אז הוא שולף את ה-node עם הsequence המינימלי שגדול מה sequence אצלו, ה-node הזה מכיל בתוכו המידע המזהה הבלוק המתאים שכבר התקבל אצל השרת.

חוטים אצל השרת:

1. **חוט שליחת הבלוק:** כששרת מייצר בלוק שלשליחה הוא מפעיל חוט שמבצע את Block Broadcast אל כל שרתי המערכת האחרים ומוסיף את הבלוק גם אצלו ב- Received Blocks, ולבסוף מייצר zkNode שמייצג את הבלוק.
2. **חוט הניקוי:** כששרת נקבע להיות המנהיג האחראי על הניקוי הוא מייצר חוט שכל כמה שניות מבצע הבא:
   a. מבקשת מכל השרתים ה-sequence שלהם
   b. מוצא את ה-sequence המינימלי מבין הכל נסמן אותו maxSeq
   c. מוחק מתיקיית BLOCKS/ את כל הבלוקים עם sequence קטן מ maxSeq, כי בלוקים אל כבר נכנסו ל-Blocks Chain וניתן למחוק אותם סופית.

   העבודה של חוט זה חשובה לסקיילביליות ולשמירת ה-zk נקי יחסית, ולשפר ביצועי המערכת.
3. **חוט שליחת בלוק שלא התקבל:** במקרה ששרת בודק Validity של בלוק ומוצא שהוא אינו Valid אז הוא בודק אם בלוק זה הוא במקור ההצעה שלו אז הוא נותן לו מזהה חדש ועושה עוד פעם Block Broadcast אל כל שרתי המערכת האחרים, דבר זה נעשה רק פעם אחת עבור הבלוק כלומר אנחנו נותנים עוד הזדמנות אחת לבלוק שלא התקבל בפעם הראשונה.

Zookeeper watches:

1. **NodeChildrenChanged של BLOCKS/:**

סוג איוונט זה אומר שנוסף/נמחק בלוק מהתיקייה, במקרה של קבלת איוונט זה השרת שולף את הבלוק עם ה-sequence המינימלי מהתיקייה BLOCKS/ ומנסה להכניס אותו ל- Block Chain, בהתחלה מעדכן את maxSequence אצלו ואז בודק את ה-Validity של הבלוק, אם הוא Valid מבחינת sequence ומבחינה טרנזקציות אז הוא מוכנס ל- Block Chain אחרת הוא נזרק.

2. **NodeChildrenChanged של OMEGA/:**

סוג איוונט זה אומר שנוסף/נמחק server מהתיקייה, במקרה של קבלת איוונט השרת בודק האם הוא אמור להיות המנהיג ואז להיות Cleaner, גם כן השרת מעדכן את רשימת שרתי המערכת לפי השרתים הקיימים בתוך תיקיית OMEGA/.

FD וטיפול בנפילות:

במקרה של נפילת שרת יתקבל watch על שינוי ב- OMEGA/, אז השרת מעדכן אצלו את רשימת השרתים החיים, בודק האם הוא בעל המזהה הכי קטן מבין ה-nodes ב- OMEGA/, אם הוא רואה שהוא הפך למנהיג החדש אז זה אומר שהוא Cleaner וצריך להפעיל את החוט שמטפל בניקוי בלוקים.
עכשיו כשהשרת נופל המערכת ממשיכה לעבוד, והנכונות של המנגנון לא נפגעת בגלל נכונות ה FD אבל כדי לשפר את ביצועי המערכת צריך לעשות כמה דברים :
כל שרת מעדכן אצלו את רשימת הבלוקים שהתקבלו ממנו ב- Received Blocks לפי מה שכבר הספיק להגיע לתיקייה BLOCKS/, כלומר אם שרת נפל לא הספיק לעשות **atomic Broadcast** לבלוק מסויים ולפרסם אותו ב zookeeper זורקים הבלוק הזה מ- Received Blocks.

Blockchain algorithm:

האלגוריתם שלנו ממש את ה- **leader free** שבו כל server שמעוניין להציע בלוק הוא שלוח אותו ישירות לכל השרתים האחרים במערכת, וסדר הכנסת הבלוקים נקבעת באופן נפרד בשלב מאוחר בעזרת zk, האלגוריתם שלנו נכון כי הוא מקיים את שלושת התכונות:

1. **Validity**: בלוק נכנס ל-Block Chain אם כל הטרנזקציות בו הם VALID.

2. **Agreement**: הבלוק שמוכנס ל-Block Chain הוא תמיד נבחר לפי ה-node עם ה-sequence העוקב שנמצא ב-zk בתיקיית BLOCK/, לכן כל השרתים יראו את הבלוקים בדיוק באותו סדר כי הם עוברים עליהם כפי ה-zk קובע וזה סדר חד משמעי,לפי האלגוריתם שלנו בלוק שמפורסם ב-zk רק אם הוא כבר נשלח לכל שרתי המערכת ונמצא ב-Received blocks, ומכיוון שכולם מחליטים עליו אם הוא valid באותו אופן אז כולם יחליטו אותה החלטה ולכן הוא יכנס בשרשרת אצל כולם.

3. **Termination**: מאופן פעולת המערכת שרת S שמציע בלוק B מגיע לכולם ומפורסם ב zookeeper ומכיוון שכל שרת לוקח בלוקים אחד אחרי השני מה zookeeper בצורה עוקבת אז כל אחד יגיע לבלוק B ומכיוון שהוא valid block אז הם יכניסו אותו לרשימה שלהם.

Adaptive Batching:

הצורך במנגנון כזה בא מהעובדה שגודל בלוק משפיע על התפוקה של המערכת, מכיוון שבלוק גדול מדי יכול לקחת זמן רב להעביר אותו ברשת לשאר השרתים.
ואם הבלוק קטן אז נצטרך להסכים על יותר בלוקים עם שאר השרתים ולבצע יותר consensus בין השרתים שעולה זמן יותר.
ולכן החלטנו על מנגנון לפתור ה trade off הזה ע״י התאמת גודל הבלוק לעומס הרשת באופן הבא:
ההחלטה לשלוח הבלוק עם הטרנזקציות שיש אצל השרת נקבעה ע״י שימוש בשני פרמטרים, **threshold** שמסמן הגודל שממנו אפשר לשלוח בלוק, וגבול של **שניה**.
כאשר השרת ממתין שניה אחת, ואם יש לו מה לשלוח הוא שולח את הבלוק, אחרת אם הגיעה כמות הטרנזקציות המתקבלות אצל השרת לגודל ה threshold שולחים אותם בבלוק.
כאשר מגיעים לגודל ה- threshold לפני שעוברת שניה אחת השרת מכפיל את ה- threshold פי 2. או מוסיף עליו 100 אם הוא גדול מ 1500, כך שבהתחלה ההגדלה תהיה בצורה מהירה (Exp) ועם הזמן ההגדלה תהיה איטית (קבועה) שלא נגדיל הבלוק יותר מדי ולא נעמיס הרשת.
אחרת אם מגיעים לשניה לפני ה- threshold אנחנו מחלקים אותו ב- 2.
וכך במנגנון הזה השרת מנסה תמיד להגיע לגודל ה- threshold שהוא הגודל האידיאלי לעומס הרשת הנוכחי, ומנסה להתאים אותו, כך שאם הוא היה גדול מדי אז תעבור שניה לפני שנגיע ל threshold הזה ולכן יש צורך להקטין אותו, אחרת אם מגיעים אליו לפני שעוברת שניה אז הוא קטן מדי ולכן יש לעדכן אותו להיות יותר גדול.

הסימולציה של ה adaptive batching:

בעומס נמוך: בתהליך כל לקוח שולח לשרת טרנזקציה בודדת.
בסימולציה זאת רואים שהבלוקים נשלחים אחרי שניה ומתקבלים אצל כל שרת ב blocks chain שלו.

בעומס גבוהה: כל לקוח שולח 10000 טרנזקציות לשרת המקושר אליו.
בסימולציה זאת רואים שגודל הבלוק עולה בצורה דרמטית בהתחלה, וממשיך לגדול כל הזמן במקום לקטון כי הרשת עמוסה, בניגוד לציפיות שלנו שזה יגדל פי 2 כל פעם בהתחלה ואז באופן קבוע אחר כך ויקטן כשהרשת עמוסה.
הסיבה מאחורי זה היתה שהשרת מבצע שליחת הבלוק לשאר השרתים בצורה סדרתית, הכוונה היא שכאשר הוא מסיים שליחת הבלוקח, הוא הולך ומסתכל על הטרנזקציות שהגיעו אליו, אבל בזמן שליחת הבלוק הקודם הגיע מספר רב של טרנזקציות ולכן הוא ישלח עכשיו מספר גדול של טרנזקציות של בלוקים ללא תלות בעומס הרשת, אלה כי השרת היה עסוק בלשלוח בלוק.
לכן החלטנו למקבל את התהליך של שליחת הבלוק, כך שבכל פעם שרוצים לשלוח בלוק לשאר השרתים מפעילים חוט שמבצע זה, כך שהשרת יחזור להסתכל על הטרנזקציות שמגיעות אליו ולבדוק מתי לשלוח אותם.

דבר נוסף לשים לב אליו,שהשרת אינו פשוט יושן שניה ואז כשהוא מתעורר בודק האם הגיע לגודל ה threshold, אלה שהוא מתעורר כל 1 msec לבדוק כמות הטרנזקציות שהגיעו אליו,וכך למעשה מתעורר 1000 פעם בשניה לבדוק את זה, וכך הוא יכול להחליט למה הוא הגיע קודם,לשניה או ל threshold ולעדכן בהתאם.

הרמת המערכת עם Docker

פשוט להריץ: "docker-compose -f servers_stack.yml up"

מבנה ה- stack:

המערכת בנויה מ 3 שרתים של zookeeper ומספר קבוע של שרתי הבנק ומספר קבוע של לקוחות.
רצינו להרים את כולם באמצעות docker, כך שבפקודה אחת המערכת תעלה ותתחיל לעבוד.
לצורך כך הכנו לכל שרת image שנריץ אותה בתוך container.
עבור שרתי ה zookeeper יש את ה images מה docker hub.
עבור שרתי הבנק, כתבנו השרת בשפת java, קמפנלו אותו לקבל קובץ exe שהרצנו בתוך container שמבוסס על image שמריץ JVM.
כנ״ל לכל לקוח שרוצה להצטרף יש את הקובץ ה exe שרץ בתוך container משלו.
כתבנו yml file שמגדיר המערכת הכוללת ה services שהזכרנו למעלה.
ובפקודה אחת אנחנו מעלים אותם ע״י ה swarm manager.
ה swarm manager מנהל את ה services בתוך ה stack שלנו והוא דואג להתחיל מחדש שרתי zookeeper שנופלים, כמו שציינו לו ב yml file.

- לתת ל node הנוכחי להיות ה swarm manager ע״י : " *swarm manager init"*
- *צריך לבנות פעם אחת בהתחלה את ה-image של שרת בנק שכתבנו ע"י הפקודה:*

  “docker build -t bankserver . "
- *להפעיל לקוח עם קובץ exe שנמצא בתיקיית bin שיצרנו ע"י הפקודה:*

  "./transactions-client localhost:5556"

  הפקודה מקבלת את ה-ip של השרת המשרת למשל:

  localhost:5555,localhost:5556,localhost:5557

Docker Visualizer:

מראה את המערכת שלנו רצה, יש לנו 3 שרתי zookeeper שכל אחד על container נפרד, ורואים את שלושת השרתים של הבנק שמשרתים את הלקוחות, גם כן כל אחד רץ על container נפרד.