אלגוריתמים מבוזרים - 67520

21 בינואר 2013

מרצה: דני דולב

איני לוקחת אחריות על מה שכתוב כאן, so tread lightly
אין המרצה או המתרגל קשורים לסיכום זה בשום דרך.
הערות יתקבלו בברכה - noga.rotman@gmail.com. אהבתם? יש עוד!
http://bit.ly/integrali

תוכן עניינים

0.0.1 מנהלות

יתכן והמרצה יצטרך לסוע פעמיים לבריסל במהלך הסמסטר, ונצטרך לעשות השלמות.

כמו כן, במידה וגודל הכיתה ישאר כפי שהוא (פי 3 מהרגיל בדר"כ), הפורמט ישתנה. למרצה חשוב להטמיע את הדרך להתמודד עם בעיה לבין ללמד פתרונות מסויימים, שכן תחום זה הוא תחום דינמי.

מה הכוונה? ציון הקורס בעבר התבסס על תרגילים (5-6) בזוגות במהלך הסמסטר, השנה תהיה בחינת סיום. עם זאת, מי שפתר את כל התרגילים לא תהיה לו בעיה. אולי תהיה בחינת בית, שוב, וריאנטים של התרגילים.

במהלך הקורס ילוו אותנו שלושה ספרים:

N. Lynch

H. Attiya & J. Welch

S. Dolev

בספר הראשון המחברת פיתחה מודל פורמלי שהוא כמו אסמבלי, ולכן פחות נשתמש בו. בספר השני הפרוטוקולים יותר אינטואיטיביים. בספר השלישי יש נושא מסויים שניגע בו בהמשך.

0.1 מבוא

מערכות מבוזרות הם מספר אלמנטים המתקשרים ביניהם כדי לבצע פעולות מסויימות. היום מדובר ברשתות גדולות מאוד. כל אלמנט יכול לפעול באופן כלשהוא עצמאית, אך ישנן משימות שיש בהן צורך בשיתוף פעולה.

ב-parallel כל ה-cpu-אים עובדים על אותה מטרה, ומנסים להקטין את זמן הפעולה. במערכות מבוזרות האלמנטים יכולים לבצע כמה פעולות עצמאיות בו זמנית, לכן הבעיה היא שונה (ניתן להתייחס לבעיה שלנו כתת בעיה של הבעיה הראשונה, אך לא נעשה זאת הפעם).

ישנו מגוון רחב מאוד של מודלים אפשריים, ולא נכסה בקורס את כולם. נציג חתכים אפשריים במודלים שונים, ונראה את הפתרונות.

סוגי מודלים:

1. סינכרוניים - ניתן להגדיר פאזות\סבבים, בכל סבב המחשבים יכולים להספיק להחליף ביניהם הודעות. במערכות שונות מידת הסינכרון יכולה להשתנות (ננו-שניות, מאיות-שניות, וכו').

2. אסינכרוניים - מקצב התקדמות במחשב שלי אי אפשר להקיש שום דבר על מחשב אחר במערכת.

   שני המודלים הללו הם מאוד קיצוניים, ומאפשרים לפתור את הבעיה ללא תלות במה שקורה ברשת: יתכן ומספר הודעות (קטן) יגיעו מאוד מאוד מאוחר. במודל סינכרוני קובעים תקופת זמן מסויימת לקבלת הודעות, כל מה שמתקבל לאחר מכן יחשב כטעות. במודל אסינכוני מתייחסים לכל ההודעות המגיעות, בכל זמן שהוא, כנכונות, ומטפלים בהם.

3. סמי-סינכרוני - יש חסם על הזמן בו הודעות יכולות להגיע. חסם זה "מספיק טוב" ליציבות המערכת.

מה לגבי ההנחות לגבי הרשת?

1. סינכרון - ההודעה תגיע לנמען או לנמענים תוך פרק זמן מסויים קבוע בכל מצב שהוא.

2. אסינכרונית - "כמו דואר ישראל: שולחים, ומתישהו זה יגיע".

3. מבנה רשת התקשורת, למשל, הרשת היא גרף מלא. לעיתים המבנה מהווה תפקיד מהותי בתפקוד הרשת.

לגבי הרשת, נניח כמעט ללא יוצא מן הכלל כי הרשת אמינה, כלומר כי היא אינה מאבדת הודעות ואינה משבשת אותן. אי שיבוש - כי אפשר להוסיף מספיק תיקון שגיאות לקוד. נוח להניח שהרשת לא מאבדת הודעות: אפשר להוסיף קאונטרים, ואז אפשר לסמלץ כאילו הרשת לא מאבדת הודעות. במציאות אין זה המצב ויש להתמודד אם המצבים הללו.
לגבי המעבדים, כאן נאתגר את עצמנו עם מודלים מעניינים יותר של התנהגות מעבדים:

1. אמינים - עושים את מה שאנחנו אומרים להם לעשות באמינות מלאה.

2. מודלי שגיאות שונים:

(א) fail-stop (או crash)[1] - המעבד מתפקד כמו שהוא מתבקש לעשות עד לנקודה מסויימת עד שהוא נופל, והחל מאותה נקודה הוא לא יתפקד כמו שצריך עד סוף הפעולה.

(ב) omission - לעיתים נפריד בין input ל-output: זהו מעבד "שהגיע לגיל גבורות" ומתפקד לא טוב, למשל לא שולח את כל ההודעות (output), או לא מטפל בכל ההודעות שהוא מקבל (input), והפעולה שלו לא צפויה. מירב אובדן ההודעות שאנחנו חווים ברשת היום נובעים מבעיה זו - הבאפר מלא ולא הכל מטופל. נדיר מאוד שהרשת מעבדת "על הקו" הודעות. זה פחות נכון לרשתות אלחוטיות, אבל נתעלם מהאספקט הזה.

(ג) timing - השעון יכול להשתנות כרצונו, או ה"יריב" יכול לגרום לו לעשות דברים בזמנים לא נכונים (למשל בעיית סנכרון השעונים אותה נראה בהמשך).

(ד) ביזנטית - ה"יריב" יכול לגרום להתנהגות מאוד לא צפויה. הגיעו לבעיה הזו כאשר התחילו לשלוח חלליות בחלל, השאלה היתה, האם אפשר לבנות מערכת שתוכל לתמודד עם **כל** שגיאה אפשרית (שכן לא ניתן להגיע לתקן שגיאות בחללית בעודה בחלל).

(ה) יצוב עצמי $(self-stabilizing)$ - גם מודל זה הוצא ע"י נאסא בשנות השמונים - החללית יכולה לעבור באיזורים עם קרינות שונות שיכולות לשבש תאים בזיכרון. במודל זה איפשהוא אנחנו חשופים לשגיאת ברת-חלוף שתשבש את כל התאים בזיכרון, והבעיה היא התייצבות מטעות שכזו, אם התאים משתבשים לכל ערך כלשהוא - האלגוריתם עצמו צרוב במקום כלשהוא, לכן גם אליו לא תהיה גישה. אם יהיה זמן, נלמד על שילוב של בעיה זו עם הבעיה הקודמת: יום הדין האולטימטיבי. עד לפי כמה שנים הפתרון היה סופר-אקספוננציאלי, לכן הוא לא היה מעשי. בעשור האחרון הצליחו להגיע למספר פתרונות מאוד יעילים להבעיה הזו, ובמידה ונגיע נציג את חלקם.

פרמטר נוסף הוא תקשורת - מפרידים בין 3 מודלים עיקריים:

[1] יש וריאנטים בספרות אך נתעלם מהם בקורס זה.

1. זיכרון משותף - מניחים שלכל המחשבים יש זיכרון שכשמישהו כותב אליו, כולם יכולים לגשת אליו. היום, למשל ב־multi-core יש יישום של מודל כזה.

2. שליחת הודעות - המעבד יכול לשלוח הודעות לכל שכניו (ומבנה הרשת קובע את השכנים). את רב המאמץ במהלך הקורס נשקיע במודל זה.

3. Broadcast - יותר דומה במובן מסויים לרשת אלחוטית מקומית: אם אחד מדבר כולם שומעים.

אפשר לסמלץ מודל אחד במודל שני, כלומר לכאורה אין בזה עניין גדול. המודל נעשה מעניין כאשר משלבים אותו עם פרמטרים נוספים.
פרמטר נוסף הוא מדדי סיבוכיות - חלקם יותר רלוונטים בבעיה אחת, ופחות מאחרת. למשל:

- מספר הודעות.
- יחס בין טובים לרעים.
- זמן סיום (מספר הסבבים).
- זמן התכנסות (למצב טוב או סביר).

יש trade-off בין המדדים השונים לבעיות ספציפיות.
הוצג כאן אוסף גדול של מודלים שונים, ולא נוכל לדון בכולם. מה שנעשה במהלך הקורס - המרצה יבחר בעיות שבמודלים ספציפיים יש בהם בעיה מיוחדת שיש להתייחס אליה. המטרה בבחירת הבעיות היא אינה לומר שנכסה את הכל, אלא נציף את הכלים המחשבתיים או הטכניים שצריך כדי להתגבר על בעיות מטיפוס מסויים שיאפיינו בעיות שניתקל בהם במציאות.
לא נטפל במהלך הקורס בכח החישובי של המחשבים השונים, ולכן הם יתאימו למערכות שיש בהן גם מחשבים רגילים, גם אייפודים, גם טלפונים ניידים, וכו'. החישוב עצמו יהיה מאמץ זניח (לכן לא נציג בעיות חישוב קשות). אי הוודאות לגבי המצב במקומות שונים ברשת תלווה אותנו הרבה - אפילו במודל הכי חזק אנו יודעים רק מה קרה עד עכשיו, ולא נוכל להשליך על העתיד (אם לא נקבל הודעה נדע שמשהו נפל, אבל לא נדע עד שיעבור פרק הזמן הזה).
בתחום זה יותר אלגוריתמים התפרסמו והתבררו כשגויים יותר מכל תחום אחר, לכן זהו תחום מאוד מסובך, ואין צורך לפחוד משאילת שאלות, ויתרה מזו, המרצה מעודד לשאול שאלות.

1 בניית עץ פורש מינימלי, או בחירת מנהיג

נתחיל עם בעיה שעל פניה נראית פשוטה, אך הפתרון בסופו של דבר יהיה מורכב. בדר"כ תרגיל הבית הראשון הוא כתיבה "נקיה" של מה שעשינו בכיתה, והוא נבדק בדר"כ רק בסוף הסמסטר (כי לוקח למרצה הכי הרבה זמן לבדוק אותו).

הנחות המודל

תחילה ננית כי:

- אסינכרוניות מלאה.

- אמינות מלאה - הודעות לא הולכות לאיבוד, והמעבדים אמינים.

זהו המודל הכי פשוט שאפשר.

1.0.1 שקילות ההגדרות

נביט לרגע על וריאנט של הבעיה: נניח שקיים עץ פורש מינימלי, וכעת נבחר מנהיג.
קיים עץ פורש מינימלי - כל אחד יודע מי השכנים שלו, ואין מעגלים.
תחילה, האם המעבדים הם זהים לחלוטין, או שיש להם id? במהלך הקורס נניח שלכל מחשב יש id ייחודי. ניתן גם לדון בבעיה ללא האלמנט הזה, ואז נכנסת בעיה של זיהוי מוחלט של מחשב, אך לא נדון בכך במסגרת הקורס. לעיתים נניח שיודעים את ה־id מראש, לעיתים לא. כאן נניח שלא יודעים, רק יודעים את הקשתות בעזרתן מדברים עם השכנים.
במקרה זה האלגוריתם מאוד פשוט: עלה יודע שהוא עלה (יש לו שכן אחד בלבד). עלה שיתעורר ישלח הודעה לשכנו, נגיד עם ה־id שלו. שאר המחשבים יחכו עד שיקבלו מספר הודעות שיהווה את מספר שכניו פחות אחד. מה האפשרויות להתרחשות ברשת?

- כשמחשב מסויים יתעורר "תיבת ההודעות" שלו תהיה מלאה, כלומר קיבל הודעות מכל שכניו, ואז הוא יכריז על עצמו כמנהיג, וזה מספיק לפתרון הבעיה שלנו.
- זוג קודקודים שכל אחד מהם שלח הודעה לחברו, ואז יהיו זוג טוענים לכתר. כיצד תפתר הדילמה? מי ששלח את ה־id הגבוה ביותר, למשל, יכריז על עצמו כמנהיג.

לכן אם יש עץ פורש (אפילו לא חייבים מינימלי - לא השתמשנו בתכונה הזו בפתרון שלנו) במודל שלנו, אפשר לבחור מנהיג. אנו לא יודעים לכמת מבחינת זמן, אך יודעים שיגיע הרגע, וזה מה שחשוב לנו. לעיתים אפשר להציב חסמים בהנתן מדדים לשליחת הודעות, עם זאת, כרגע זה לא יעניין אותנו.
בכיוון השני, נניח כי קיים מנהיג, ונרצה למצוא עץ פורש מינימלי. יש לציין מינימלי על מה - אפשר לציין משקל שרירותי כלשהוא לכל קשת בעת הגדרת הבעיה, אבל כאן נניח כי לכל קשת יש זוג סדור של ה־id של הקודקודים, והמשקל של הקשת יהיה המשקל של ה־id המינימלי בזוג זה, ועל הזוגות מוכל סדר לקסיקוגרפי, כלומר $\{1,2\} < \{1,3\}$.
קיים מנהיג: כל מחשב יודע אם הוא עצמו מנהיג או לא.
רעיונות: המנהיג מוצא את הקשת המינימלית ממנו ואז יוצא ממנה, או עובר ב־BFS ומתקן.
הבעיה: אם לא היינו דורשים את המינימליות הספציפית, המנהיג יכול היה לשלוח לכל שכניו, שישלחו לשכנים שלהם, וכו'. אז קודקוד שמקבל מכמה שכנים יכולים לבחור את "אביו" לפי פרמטר כלשהוא: ה־id הנמוך ביותר, זמן ההגעה, וכו'. כך כל קודקוד שולח לכל שכניו, פעם אחת נבחר "אב", ואם נסתכל על שרשרת האבות, האם נקבל עץ פורש? לאו דווקא! יש להוסיף לפרוטוקול:

- חשוב להוסיף לפרוטוקול כי מי שכבר בחר אב שיתעלם מהודעות נוספות.
- המנהיג לא יבחר אב.
- הניואנס: ההנחה היא שקודקוד בוחר אב לפני ששולח לכל שכניו, ועל כן אין מעגל.

כך נקבל עץ פורש, לאו דווקא מינימלי.
בפרוטוקול זה מספר ההודעות שנשלחות הוא פעמיים מספר הקשתות בגרף. מאוחר יותר נעשה וריאנט לפרוטוקול זה כדי למצוא עץ פורש מינימלי.
מהניתוח זה, הבעיות אכן חופפות.

1.1 וריאנט ראשון: כל מחשב יודע את רשימת שכניו

נשים לב כי כל מחשב צריך להיות איפשהוא בעץ.

טענה 1.1 אם קודקוד מחובר בעץ פורש מינימלי, הקשת המקשרת אותו ל־id המינימלי חייבת להיות חלק מהעץ.

הטענה ברורה מהאלגוריתם החמדן שראינו למציאת עץ פורש:
נניח בשלילה כי נמצא עץ פורש מינימלי, והקשת המינימלית המקשרת קודקוד מסויים אינה בעץ זה. הוספה של קשת זו תיצור מעגל. הורדה של הקשת המקורית שקישרה את הקודקוד הזה לא תפגע בקישוריות העץ, ורק תוזיל את העלות, בסתירה שהעץ היה פורש מינימלי.
וריאנט של טענה זו היא הנחה של כל מחשב במערכת זו: "הקשת הכי "זולה" היוצאת ממני תמיד בעץ פורש מינימלי".
יש פתרון טריוואלי לבעיה ־ כולם ישלחו לכולם, מחשב אחד יאסוף ויבחר בהתאם. אבל פתרון זה יקר מאוד, ואנו צריכים לצמצם את עלותו.

הפרוטוקול הראשוני:

1. בחר את הקשת המינימלית היוצאת ממך ושלח הודעה לשכנך.

   לאחר שלב זה, נוצר יער של רכיבים מכוונים, המובילים למעגל בעל שני קודקודים (לא יתכן מעגל גדול מכך מהעובדה ששלחנו הודעה מינימלית). נשים לב כי יתכן ולא כל המעבדים התעוררו. תת־הגרף המינימלי הוא שני קודקודים. למקרה של שני קודקודים במעגל נקרא בהמשך "שידוך מוצלח" ־ אחד מהשניים יהיה מנהיג של רכיב הקשירות הזה. על הצעד הזה נחזור כמה פעמים, אבל לא ישירות, ונצטרך למצוא דרך לקשר בין רכיבי הקשירות שיתקבלו. בכל רכיב קשירות כלשהוא, עד שמתקבל ה"שידוך המוצלח", אנו יודעים שלא נמצא עדיין ה"לב" שלו (=השידוך המוצלח, שאנו יודעים שקיים בכל רכיב קשירות), ויש לחכות שיתגלה. כשהוא מתגלה, אנו נדע מי המנהיג המקומי ־ הקודקוד בעל ה־id הנמוך ביותר מבין השניים במעגל.

2. המנהיג ישלח הודעה שהוא מנהיג למעבדים שהתעוררו אשר תכלול שם מסויים לרכיב הקשירות ־ idk, ואז שאר המעבדים ידעו שהפאזה הראשונה נגמרה (המנהיג צריך לדעת שכולם קיבלו את ההודעה ־ אפשר לחכות לאישור קבלה (ack) כדי לוודא זאת).

   כעת, יש למצוא את הקשת המינימלית המקשרת בין המנהיג של רכיב הקשירות (במידה ונמצא) לרכיב קשירות אחר. כיצד עושים זאת? כל מעבד ישלח את הקשת המינימלית שאינה בתוך רכיב הקשירות וישלח אותה הלאה, עד שיגיע למנהיג (כמובן שכל קודקוד יחכה לתשובה מכל בניו לפני שיעביר את הבחירה הלאה). אבל, איך יודעים מי נמצא בתוך רכיב הקשירות?

3. כל מעבד ישלח "הודעת גישוש" לקודקוד הבא בתור ברשימת שכניו עם ה־idk שלו ומחכה לתשובה ממנו. אז המעבד המקבל בקבלו הודעת גישוש, ישלח הודעה בחזרה באם הוא ברכיב או לאו.

   מה קורה לקודקודים שברכיב הקשירות, אך לא יודעים זאת עדיין כי לא התעוררו וקיבלו את ההודעה? כשהקודקוד מתעורר הוא יראה את הודעת הגישוש ששלחנו לו, אבל צריך לוודא שסדר ההודעות יהיה נכון! כלומר, שבעת התעוררות קודם כל מעבד שמתעורר ישלח את ההודעה הראשונית. לאחר מכן עדיין הקודקוד לא יכול לטפל בהודעת הגישוש, שכן הוא עדיין לא יודע מיהו המנהיג. על כן בפרוטוקול יש

אי וודאות לגבי רכיב הקשירות בו המעבד נמצא, ובשלב זה אי אפשר לענות להודעות גישוש מסויימות (ונראה את המשמעות של "מסויימות" בהמשך), כי יכול להיות שמענה שכזה יצור מעגל ברכיב.

על כן, אין צורך בחלק משלב 2: מספיק שהמנהיג יפיץ הודעה ברכיב הקשירות שתאמר "זהו שם רכיב הקשירות", ובקשה למציאת קשת מינימלית היוצאת מהרכיב (אין צורך בקבלת אישור מכולם). כאמור, בעת קבלת הודעת גישוש, מחכים עד שיש שיוך לרכיב קשירות.

נניח כי שלב זה נגמר, כלומר המנהיג הצליח למצוא את הקשת המינימלית היוצאת מהרכיב לרכיב קשירות אחר. זו חייבת להיות הקשת המינימלית, כולל את הרכיבים שעדיין לא התעוררו. וזו אכן הקשת המינימלית, כי אם היתה יוצאת קשת יותר זולה החוצה מקודקוד "רדום" שכזה s, המקושר לקודקוד ברכיב הקשירות q, שבשלילה מצא יציאה מהרכיב בעזרת קישור לקודקוד חיצוני w, אזי הקשת בין s ל־q זולה יותר, ולכן q היה מנסה קודם כל לשלוח הודעת גישוש מ־s. s רדום ועל כן לא ענה, ולכן q עדיין מחכה לתשובה, ולכן לא יבחר את הקשת המחברת ל־w.

אזי בסוף התהליך, המנהיג מחכה לאסוף ההודעות מכל בניו, ואז יכול לקבוע סופית מיהי הקשת המינימלית היוצאת מרכיב הקשירות.

4. משנים את הכיוון של העץ שיכוון לקודקוד שיוצא החוצה, ושולחים הודעת שידוך לקודקוד המתאים.

כעת כל רכיב הקשירות יראה כפי שקודקוד בודד נראה בפאזה הקודמת ־ יודעים באיזה רכיב קשירות הקודקודים היו, אבל כעת לא ידוע מהו רכיב הקשירות החדש. אזי, רכיב הקשירות הוא כמו "מטה־קודקוד" בפאזה הקודמת.

מה קורה בצד השני? נניח כי הקשת הזולה ביותר היא בין q ברכיב הקשירות "שלנו", לבין w, קודקוד ברכיב קשירות אחר. מקרה ראשון: הודעת השידוך מ־q מגיעה לפני ש־w סיים את הפעולה שלו ־ במקרה זה w מפיץ לכולם "אנחנו חלק מרכיב הקשירות החדש". מקרה שני: w כבר החזיר תשובה למנהיג, כלומר הוא בעת חיפוש. אם הקשת שהתקבלה היא הקשת ל־q, הם יתחברו. אחרת, w יעביר את בקשת החיפוש שלו ל־q.

עד כאן העלות (מספר ההודעות) הולכת בכיוון של n^2, כי יהיו הרבה בקשות עודפות במערכת. נרצה ליעל את הפרוטוקול כפי שהוצג עד כה.

כש־w קיבל <u>הודעת גישוש</u> מ־q, לפחות קונספטואלית, אם נדאג שישתדכו רק רכיבים "באותה הרמה" (באותו סדר גודל), אז נוכל ליעל את הפרוטוקול!

5/11/2012

בשבוע שעבר הצגנו פתרון בסיסי לבעיה, המתבססת על סוגי הודעות:

- הודעת גישוש ־ מטרתה לברר אם הקודקוד השני באותו רכיב קשירות.
- הודעת שידוך ־ מרכיב הקשירות שלי, הקשת ממנה יוצאת ההודעה היא הזולה ביותר היוצאת מרכיב קשירות זה החוצה, כולל הקודקודים שטרם התעוררו. זוהי נקודה מאוד עדינה.
- בתוך רכיב הקשירות היתה הודעה פנימית מופצת ע"י המנהיג של רכיב הקשירות המציינת את ה־id של הרכיב, פרמטר נוסף שעוד לא פירטנו, ובקשת חיפוש, המופצת על ידי כל הקודקודים הלאה.

ראינו שחלק מנוון של שלושת ההודעות האלו היא ההודעה שמעבד מוציא כאשר הוא מתעורר[2].

כל אחד מהקודקודים הפנימיים שערים כרגע אוסף את הקשת המינימלית ושולח אותה למנהיג, שבוחר את הקשת הזולה ביותר, והיא תהיה הזולה ביותר מכל רכיב הקשירות. הוכחנו בצורה משכנעת כי היא אכן הזולה ביותר, גם מבין הקודקודים ברכיב הקשירות שעדיין לא התעוררו.

נשים לב כי כאן קיימת "נקודת סינכרון" - לאחר בקשת החיפוש המנהיג מחכה לתשובות ולא שולח בקשת שידוך לפני שקיבל תשובות מכל הקודקודים שהתעוררו.

לאחר מציאת הקשת, המנהיג עובר להיות השכן של הקשת הזולה ביותר היוצאת מהרכיב החוצה. בתהליך זה הכיוון של הקשתות במסלול זה מתהפך.

כל זמן שלא שלחנו את הקשת המינימלית שיוצאת מאיתנו החוצה, אנחנו יודעים בוודאות באיזה רכיב קשירות אנחנו נמצאים. ברגע שנשלחה, אנו לא יודעים עם נעשה שידוך או לא, ולכן אנו לא יודעים את ה־id של רכיב הקשירות (נובע מהא־סינכרוניות). עם אי הוודאות הזו נצטרך להתמודד.

רכיב הקשירות אליו מוציאים את בקשת השידוך עושה תהליך דומה, ויתכן שגם רכיב זה יוציא את הבקשה בשלב מסויים. מצב שכזה, בו נשלחו הבקשות משני הצדדים נשלח "שידוך מוצלח", ואז אחד משני הקודקודים (נגיד המינימלי מביניהם) נהיה המנהיג, ויוצאת הודעה חדשה לקודקודים שתזהה את רכיב הקשירות החדש. על כך נרחיב עוד בהמשך. נשים לב כי "שידוך מוצלח" הינה קשת יחידה בין שני רכיבים - שאר הבקשות יהיו בקשות שלא יענו.

כשהוצאנו בקשת שידוך, כולם מחכים, ומקבלים הצעות שידוך. מתישהו, כשיהיה שידוך מוצלח, תופץ הודעה "זהו השם החדש שלנו" לכל הרכיב המאוחד. כאשר תגיע כזו לקודקוד שהוציא בקשת שידוך, הוא ישלח ההודעה בהמשך לרכיב החדש שניסה לשדך.

הבעיה שהטרידה אותנו בשבוע שעבר היא הסיבוכיות. עלול להיות מקרה שכל העבודה הלינארית הזו (משלוח כל ההודעות בדרך לשידוך מוצלח) תסתיים בהוספה של קודקוד או שניים בלבד לרכיב מאוד גדול: תשלום לינארי בגודל הקלאסטר עבור מעט מאוד קודקודים, ואז המחיר עם הזמן נהיה גדול מאוד.

האבחנה משיעור שעבר אמרה: אם השידוכים המוצלחים יהיו בין קלאסטרים בני אותו הגודל, אזי מספר השידוכים שעוברים במהלך חיפוש העץ הפורש המינימלי היה הופך ללוגריתמי במקום לינארי. זוהי המטרה שלנו.

איזו דרגת חופש קיימת בפתרון שלא ניצלנו אותה עדיין? אם היינו מבטיחים שלאחר השידוך ה־k הגודל של הקלאסטר יהיה לפחות 2^k, וזו אינווריאנטה שתוכל להשמר, אז יכולנו להוסיף ל־id את ה־k הזה להודעה הפנימית. עבור הקלאסטר הגדול שהשתדך, ה־k יעלה ב־1, אבל קודקודים אחרים יכולים "לקפוץ" ב־k, כלומר לא לעבור את כולם בצורה לינארית. איך ננצל את האינפורמציה הזו? בהודעת הגישוש, נצרף את הזוג $\langle id, k\rangle$. הכיוון שהוצע - בקשת הגישוש (של הרכיב הקטן) תענה רק כאשר k' של הרכיב השני שווה ל־k של הרכיב המקורי, כלומר הרכיב המקורי "נתקע" ויודע מיהו. אבל, הקלאסטר הקטן חייב לגדול, ויכול להיות שבשלב מסויים, לפני השיוויון, הקשת הזולה ביותר היא זו. אם לא נענה, נגיע כאן לדדלוק.

כדי להמנע מכך, בשלב זה הוא יוציא בקשת שידוך לקודקוד של הרכיב המקורי. אזי איך נגיב ונשמור על העלות נמוכה? ישלח לרכיב המצורף (הקטן ממנו) את השם החדש כדי שימשיך לחפש, והרכיב החדש יפסול את הקשת הזו (לא תמשיך את הרכיב החוצה) וימשיך לחפש את הקשת הזולה. אם כך על כל קודקוד לשמור את ה"דור האחרון" - $\langle id, k\rangle$ שלו (אולי לא יודע בכל שלב מהו $\langle id, k\rangle$, אבל יודע את הקודם).

[2] כאמור אנו מניחים כי כל מעבד יודע את רשימת שכניו - אפשר להתעלם מהנחה זו ע"י אינישיאליזציה לפני פרוטוקול זה: שליחת הודעות לכל הקשתות, לקבל הודעות בחזרה מהן להרכיב רשימה זו, ולפני כן לא מתחילים את התהליך.

התוספת לפרוטוקול

כל קודקוד ישמור את $\langle id, k\rangle$ האחרון שלו.
בעת קבלת הודעת גישוש $\langle Id', k'\rangle$:

- אם $k' < k$, תענה שרכיב הקשירות שונה ("משחררים" את רכיב הקשירות השני כדי שיוכל להמשיך לגדול, כי אם $k' < k$ אנחנו בטוחים שרכיב הקשירות שונה).
- אם $k' = k$, $id' = id$, שלח דחיה. במקרה זה הרכיב השני חיכה עד שאחד ה־k־אים גדל, וזהו כבר אותו רכיב.
- אם $k' = k$, $id' \neq id$, ענה רכיב קשירות שונה.
- אם $k' > k$, חכה.

כעת אנו יודעים כי כאשר הוצאנו את בקשת השידוך, הרכיב אליו יוצאת הבקשה, ה־k' שלו לפחות k או גדול ־ אחרת הוא לא היה עונה על בקשת הגישוש, והרכיב שלנו היה ממתין.
טיפלנו במקרה $k' < k$ וכן במקרה $k < k'$. מה קורה אם $k' = k$?
נדון במקרה הבא: יצאה הודעת שידוך, קודקוד מסויים התעדכן, ושכן כלשהוא של קודקוד זה עדיין לא קיבל את העדכון. אם הקודקוד הנוכחי ישלח אליו בקשה, ניפול שוב למקרה $k' > k$, ולכן נחכה עד שהוא יתעדכן.
לכן, במקרה $k' = k, id' \neq id$ אנו יכולים להיות בטוחים שרכיב הקשירות שונה (שכן מהניתוח מעלה, אם הרכיב זהה בעת עדכון k נקבל שיוויון של שני הפרמטרים).

בקבלך הודעת שידוך:

- אם אתה בשלב גישוש, הצף אליו את בקשת החיפוש שקיבלת.
- אם אתה אחרי שלב הגישוש (כלומר כבר שלחנו למנהיג את הצלע הזולה ביותר היוצאת ממני), הכנס אותו לרשימת הבנים וחכה (שכן עדיין לא יודעים מהו השם החדש).
- אם הבקשה הגיעה על אותה קשת בה אני ביקשתי, זהו "שידוך מוצלח".

לאחר כל עדכון קודקוד יעבור על כל ההודעות שלו (גישוש ושידוך) ויבדוק אם יכול לענות למישהו שמחכה.

סיבוכיות

בסופו של דבר, כל קודקוד עובר על רשימת שכניו. חלק מהשכנים הוא בודק כמה פעמים, חלקם פעם אחד. לכן בפרוטוקול הזה תהיה סיבוכיות של $O(|E| + ?)$.
יש $log(n)$ שידוכים מוצלחים, כאשר בכל שידוך מוצלח העלות היא גודל רכיב הקשירות (הודעות העדכון). עם זאת, בשלבי הגישוש אנחנו עוברים על חלק מהקודקודים כמה פעמים, האם העלות $nlog(n)$ כוללת את הודעות אלו? דהיינו, נרצה לוודא שכל הודעת גישוש מכוסה בספירה הזו.
נשים לב כי כל הגישושים הלא מוצלחים (שלא הסתיימו בשידוך) נספרים בתוך ההודעות הלינאריות ־ ההודעות שפוסלות קשתות, לכן מכוסות תחת ה־$|E|$. לכל שידוך מוצלח, יש קשת אחד שעליה ענינו חזרה כקשת הזולה ביותר שיצאה מאיתנו ־ אלו נספרות ב־$nlog(n)$. לכן בסה"כ הסיבוכיות היא:

$$O(E + nlogn)$$

1.1.1 תרגיל ראשון

לנסות לנסח בצורה נקיה את הפרוטוקול שכתבנו בשעות הראשונות. עולה השאלה, איך כותבים פרוטוקול ברשת מבוזרת? ישנם מספר פרוטוקולים.

הפסודוקוד צריך להכיל מה כל קודקוד שולח, ומה הוא עונה לכל הודעה. כמו כן, יש לכתוב את הנימוקים העיקריים מדוע הוא נכון.

הצעה: לעיין בספרים. השיטה של ננסי לטעם המרצה מאוד טרחנית, אבל אם זה עובד עבור מישהו, בשמחה. מנסיון, השיטות של חגית ושלמי (?) יותר אינטואיטיביים.

כמו כן, ישנם שני כינוסים - $PODC$, $DISC$: לעבור עליהם ולהחליט על פרוטוקול "שמדבר אלינו".

הגשות רק במייל - dolev@cs.huji.ac.il. ככל שדוחים את הכתיבה, יקח יותר זמן! מועד הגשה ביום שני הבא.

1.2 וריאנט שני לבעיה עם Self-Stabilization

מעבדים תקינים, א־סינכרוניות, אבל המצב ההתחלתי אינו כמו במקרה הקודם: המודל מניח שבמצב ההתחלתי המידע שרירותי, וכן אינו יודע שהתחיל כרגע.

תחילה נציג את הבעיה הכללית, לאחר מכן נכלול הנחות נוספות שיעזרו לנו לפתור את הבעיה, בהתחלה ללא דרישה מסויימת על הסיבוכיות.

לכל מעבד יש מצב התחלתי כלשהוא שלא ידוע לנו עליו כלום: id ופרמטרים שונים שלא נקבע כרגע מהם, ונרצה להגיע למציאת עץ פורש (כרגע אין לנו צורך במינימלי) בהמשך בשלב מסויים.

נשים לב כי במצב זה ישנה הנחה כי קיים "גל" של שגיאות, ובשלב מסויים אין יותר שגיאות, כל ההודעות הישנות נמחקות מהמערכת, וכל ההודעות אמינות. כצפוי, אין אנו יודעים מתי נגמר גל זה. לשם נוחיות, נניח כי רשימת השכנים צרובה בזיכרון, כלומר לא נמחקת, וכן גם ה־id, וגם הידיעה שקודקוד מסויים הוא מנהיג.

הבעיה היא מציאת דינמיקה של פרוטוקול שכל הזמן (וכאן התובנה) מרענן את המצב שלי. מדוע? אם יש משתנה שלא מרענן את עצמו - הוא עלול להיות שגוי, לכן תמיד יש לרענן את המידע. אזי נרצה שהמנהיג כל הזמן ישלח הודעות "תזכור" לשכניו כי הוא המנהיג. מה מבטיח שה"גל" שיוצא מהמנהיג ישטוף את כולם?

הצעה: שליחת קאונטר מספר ההודעות יחד עם ההודעה "אני מנהיג", ובחירת המנהיג בעל הקאונטר הגדול ביותר. הבעיה היא שיכול להיות שעקב טעות קאונטר יהיה ממש גדול, וזה "ידפוק" את המערכת.

הצעה אחרת: שמירת קאונטר שיסמן את המרחק מהמנהיג בגרף.

אבל זה לא מספיק... במקום זה, לא נסמוך על הזיכרון שלנו, נדרוש מקודקוד לדגום את המרחק מהבנים, ונבחר ממנו את המינימלי. מה אם המרחק המינימלי הזה שגוי? לכל אחד משכניו, המספר יהיה קטן או שווה למספר זה, אזי הוא יבחר את המינימלי הזה ויוסיף לו אחד. לכן המינימום כל הזמן גדל, עד שיגיע למספר האמיתי.

19/11/2012 בשבוע הבא נוותר על העובדה שהמנהיג יודע שהוא מנהיג, וננסה לפתור את הבעיה הזו.

26/11/2012 שיעור חסר, יושלם בהקדם.

תזכורת מהשיעור הקודם:

יש מערכת שמנסה ליצור עץ פורש\מנסה למצוא מנהיג, יתכן ומאבדים\מישהו משנה את כל הערכים בזיכרון, ונרצה לוודא שהמערכת לאחר זמן קצר\סופי מתייצבת - כלומר בעץ הפורש שהפרש כל אחד יודע מי האבא שלו (אם היינו רוצים שכל אחד ידע מי הבן, זה גם אפשרי).

בהסתמך על כך שה־id צרובים, צריכים להיות 2 טיפוסי הודעות:

1. הפצה - הודעות $breadth - first$ ממי שחושב שהוא המנהיג, כאשר לבסוף המנהיג האמיתי יציף את כל המערכת. כדי לדעת מיהו האמיתי - יתקבלו 2 הודעות רציפות.

2. תחזוקה - מי אני חושב שהמנהיג, ומה מהרחק שלי ממנו.

כדי "להאמין למישהו" כאמור יש לקבל את אותה הודעת הפצה פעמיים ברצף.
נשמר אצל כל קודקוד: $Dist$ - המרחב מהמנהיג, Id - מי שאני מאמין שהמנהיג כעת, F - אבא שלי.
נסמן את ההודעה משכן p להיות:

$$\langle Dist_p, Id_p, Type \rangle$$

כאשר $Type$ הוא סוג ההודעה.
כמו כן, נסמן ב-$\overline{F}, \overline{Id}, \overline{Dist}$ להיות הערכים הכי קטנים שקיבלנו בקבוצות $F, Id, Dist$.
...

הפרוטוקול (מבולגן מאוד, יסודר מאוחר יותר)

1. אם ההודעה שקודקוד מקבל היא מהצורה $\langle Dist_p, Id_p \rangle < \langle \overline{Dist}, \overline{Id} \rangle$ בהודעת הפצה, אזי נעדכן:

$$\overline{F} := \{p\}$$
$$\overline{Dist} := Dist_p$$
$$\overline{Id} := Id_p$$

2. אם ההודעה שקודקוד מקבל היא מהצורה $\langle Dist_p, Id_p \rangle < \langle \overline{Dist}, \overline{Id} \rangle$ בהודעת תחזוקה, אז צריך לעשות ריסט (שכן בטוח שיש מישהו יותר טוב):

$$Dist := \{0\}\,,\ Id = \{v\}\,,\ F,\ \overline{F} = \phi$$

3. אם שמענו בהודעת הפצה:

$$\langle Dist_p, Id_p \rangle = \langle \overline{Dist}, \overline{Id} \rangle \wedge p \notin F$$

אזי:

$$F := F \cup \{p\}$$

4. אם מקבלים מ-$p \in F$ משהו שאינו $\overline{Dist}$, יש להוציאו מ-F.

למה הפרוטוקול עובד? נטען שבכך כיסינו את כל האפשרויות. לא כתבנו מתי שולחים את התחזוקה: מידי פעם. אם אני המנהיג, נשלח הפצה לכולם, אחרת אשלח בתחזוקה את מי שאני חושב שהוא המנהיג + $Dist + 1$.

נניח שיש רעש ברשת (יש מנהיגים פיקטיביים עם מרחקים פיקטיביים). נניח שבאותו שלב יש עדיין הודעות פיקטיביות, אז ההודעה הפיקטיבית תשתוף את הרשת פעם אחת.

אחרי הגל הזה היא לא תסתובב בלופים - כי מגדילים את ה-$Dist$ כל הזמן, וההפצה חדשה של הפיקטיבי לא תקרה כי ה-$Dist$ גדל, והמנהיג פיקטיבי ולכן לא ישלח את ההודעה עם מרחק 0. לכן בשלב מסויים כל אחד יהפוך את עצמו למנהיג וישלח הודעת הפצה, אם מקבלים $Dist$ טוב יותר - מעדכנים.

איפשהוא ברשת נמצא המנהיג האמיתי - כל עוד קיבל את בודעות פיקטיביות שמשפרות - יעדכן, וכאשר ה-$\overline{Dist}$ גדל מידי הוא יעדכן אצמו למנהיג (בסדר גודל של קוטר הגרף).

השלב זה הוא ישלח לכולם את עצמו עם $Dist = 1$ לשכניו. הודעה זו תגיע לכל הקשת בהודעות הפצה, כי היא תנצח כל הודעה פיקטיבית הקיימת ברשת - ולבסוף, יאמץ את המינימום. לאחר מכן כל הרשת מתייצבת.

איך המנגנון הזה עוזר לנו? ההודעות יכולות להגיע מכיוונים שונים - את ה-$Dist$ הכי קטן שומרים, ואת ה-$\overline{F}$.

בשלב מסוים ההודעה תגיע מאחד השכנים, ואז מתייצבים ואל מעדכנים יותר.

מפאת חוסר הזמן לא ננתח ביתר פירוט.

2 בעיית ההסכמה

תחילה, נחזור לעולם הסינכרוני.

2.1 הגדרת הבעיה

הנחות המודל רשת סינכרונית, אמינה ומלאה בה n מעבדים. אנו נדון במספר מודלי שגיאה לבעיה זו.
יש למצוא פרוטוקול אשר יעמוד בתנאים הבאים:

1. כל הטובים צריכים להסכים על אותו הערך.
2. אם יש מנהיג טוב, יש להסכים על ערכו.

   (א) במידה ואין מנהיג, אם כל הטובים התחילו עם אותו ערך התחלתי, זהו ערך ההחלטה.
3. תוך זמן סופי יגיעו להחלטה.

הערה 2.1 יש וריאנטים שונים לערך הסופי במקרה בו 2א' לא מתקיים, כאן נשתמש בדרגת חופש.

מודל השגיאה ממנו נתחיל יהיה:

2.2 בעיית ההסכמה במודל שגיאה $Fail - Stop$ במערכת סינכרונית

תזכורת - במודל זה, מחשב ש"מתקלקל" (למחשב כזה נקרא בהמשך "מחשב רע") בשלב מסויים מספיק לתפקד, ויכול להיות שלא הספיק לשלוח את כל ההודעות שהיה אמור לשלוח לפני הנפילה.
המעבדים שלא נופלים הם המעבדים הטובים.
כדי לפשט נתאר את הוריאנט עם מנהיג. הכי פשוט:

2.2.1 פרוטוקול פשוט

- פאזה 1: המנהיג שולח ערכו לכולם.
- פאזה 2: אם קיבלת ערך מהמנהיג - הסכם עליו, ושלח הערך לכולם. אם לא שמעת על ערך כלשהוא, החלט על ערך דיפולטיבי $\perp$.

פאזה במערכת סינכרונית - אם טובים שלחו בתחילת הפאזה, כל הטובים מקבלים את כל ההודעות מהטובים עד סיום הפאזה.
אבחנה: אם כשקודקוד טוב יקבל ערך בפאזה הראשונה, כל הטובים יקבלו אותו ויסכימו עליו בסוף הפאזה השניה.
האם יתכן שאף קודקוד טוב לא יקבל ערך בפאזה הראשונה? כן:

- אם המנהיג לא שלח לאף אחד.
- אם המנהיג הספיק לשלוח רק לתת קבוצה שלא מכילה אף טובים, והם שלחים לחלק מהטובים.

מה עושים? מוסיפים פאזות!

2.2.2 הרחבה של הפרוטוקול הפשוט

- פאזה 1: המנהיג שולח ערכו לכולם.
- פאזה i: אם קיבלת ערך מהמנהיג - שלח הערך לכולם.
- פאזה n: אם שמעת על ערך מהמנהיג, החלט עליו. אם לא שמעת על ערך כלשהוא, החלט על ערך דיפולטיבי $\perp$.

אבל, האם אנחנו לא מסתכלים על יותר מידי פאזות? הרי אם המנהיג טוב, אולי אין צורך בכל כך הרבה פאזות.
$n-1$ פאזות בטוח יספיקו - כי אם יש "צדיק בסדום", הוא האחרון שנותר.
נניח כי מספר המחשבים הנופלים הוא עד t. במקרה זה, יספיקו $t+1$ פאזות - אז מובטח שיש פאזה בה אף מחשב לא נפל.

2.2.3 הוכחת נכונות הפרוטוקול המורחב

נרצה להוכיח כי הפרוטוקול מגיע להסכמה.

- הפרוטווקל מסיים בזמן $t+1$, והוא סופי, על כן התנאי השלישי מתקיים.
- כל הטובים מסכימים על אותו ערך:
לאורך כל הדרך, יהיו דברים שאני אדע ואולי אחרים לא - כדי להגיע למצב שבו כל הטובים יודעים שקול להגעה להסכמה. נרצה להוכיח:

טענה 2.2 אם טוב כלשהוא החליט, אז כל שאר הטובים החליטו.

הוכחה: יש שתי אפשרויות:

- טוב החליט על $\perp$.
- טוב החליט על ערך.

מתי יש בעיה? אם יש טוב שקיבל פעם ראשונה ערך בפאזה $t+1$, ואחר שלא קיבל ערך. כדי שהערך ישרוד וישלח, הטוב שקיבל אותו היה צריך לשמוע אותו ממישהו. אורך השרשרת הוא $t+1$ (היו $t+1$ שליחות שונות), אזי לאורך $t+1$ היה חייב להיות טוב שאם שמע את הערך ישלח אותו לכולם, ולכן כזה מקרה לא יתכן. ■

2.2.4 הצעות ייעול שונות לפרוטוקול המורחב

שיפור שהוצע (נובע מהתמימות שאנחנו שולחים ערכים) - אולי אפשר לסיים בפחות פאזות אם כבר בפאזה הראשונה ישלחו "לא שמעתי" בפאזה 2 - ואז אפשר לדעת שהמנהיג רע, ואז ניתן להחליט על ערך שרירותי $\perp$ (כי אז אין צורך שהתנאי הראשון של ההסכמה יתקיים). אפשר להאמין, כי אף אחד לא ישקר בהודעה "לא שמעתי", כי במודל שגיאות זה (בהמשך נדבר על מודלי שגיאות אחרים, כאמור), רע לא יכול לעשות זאת - ברגע שנפל, נפל, ולא ישלח הודעה, ולכן כל ההודעות מגיעות ממי שטוב כרגע (אולי יפול בהמשך).

החלשה של הפרוטוקול אם יש אחד שלא שמע, והספיק לספר לחלק, אז חלק מהטובים יחשבו שהמנהיג רע, וחלק לא.

אפשר לחשוב כאן על וריאנטים - אולי צריך שכולם יחשבו שהמנהיג רע, למשל. כל הוריאנטים לגיטמיים. אולי $n-t$ מספיק.

שכלול - נוסיף פאזה שלישית - נספר כמה שלחו לי ערך, כמה שלחו שלא קיבלו, ולאור זה תתבצע החלטה. הגבול הבעייתי - האם קיים פרוטוקול משוכלל יותר שיגיע להסכמה בפחות מ-$(t+1)$ פאזות? אינטואיציה - אין כזה!

2.2.5 אין פרוטוקול אחר שיגיע להסכמה בפחות מ-$t+1$ פאזות

איך מוכיחים? נראה כי לא משנה מהו הפרוטוקול, תמיד יש ריצה בה אם הפרוטוקול עוצר עד פאזה t, אפשר לשמור שני קודקודים טובים כך שאחד הסכים על ערך, והשני לא קיבל. ופורמלית:

משפט 2.3 לכל פרוטוקול הסכמה במודל $Fail-Stop$ עם ערכים בינאריים, קיימת ריצה שלוקחת יותר מ-t פאזות להגיע להסכמה, כלומר, עד פאזה זו (כולל) קיימים שני קודקודים שאינם מסכימים על ערך.

נשים לב לנקודה עדינה: יכול להיות שבפרוטוקול יש הרבה ריצות שעוצרות לפני $t+1$ פאזות, אנו נוכיח כי קיימת ריצה כנ"ל לכל פרוטוקול. כדי להוכיח, נפנוי הידיים הקודמים שלנו לא עוזרים, וצריך להיות פורמליים, ולמדל טוב יותר את כל הנושא של התפתחות הפרוטוקול ואיך הריצות נראות.

תיאור הכלי בו נשתמש נניח בשלילה כי קיים פרוטוקול P שתמיד עוצר ב-$\geq t$ פאזות.

נביט בהיסטוריה: *איורים של היסטוריות כמשולשים שווי צלעות*

היא מתארת את כל אחת מההודעות. נביט בהבדלים בין ההיסטוריה של מעבד p למעבד p'. *איור*

נניח $t \leq n-2$. נניח כי בנוסף ל-p ישנו p' שלא ראה את ההבדל בין שתי ההיסטוריות, וכי p טוב.

אם מתקיימת היסטוריה קונסיסטנטית עם זו שאני רואה, בכולם לא יראה הבדל - הוא מחוייב להחליט ערך מסויים (במקרה של ההיסטוריות הללו - 1), ולכן גם טוב אחר חייב להחליט על אותו ערך.

הגדרה 2.4 היסטוריות הן שקולות, ונסמן $H \sim \overline{H}$, אם יש טוב שלא רואה את ההבדל ביניהן.

לפי ההנחה, p' מסיים כי ההיסטוריות שקולות. p גם חייב להסכים על אותו דבר, כי אחרת ההנחה שגויה (יכול להיות ששקול).

היסטוריה *איור H_1* קיימת כי יתכן ואחד נפל.

מסקנה 2.5 אם היסטוריות שקולות - שני קודקודים מחוייבים להחלטה על אותו הערך.

באותה מידה, אפשר לתקן את *איור H_1'* ולחזור ל-H_1, כל הזמן שמניחים כי מודל השגיאות הוא $Fail-Stop$.

נראה כי ע"י סדרה של טרנספורמציות שכזו, H_1 שקול ל-H_0.

טענה 2.6 אם 2 היסטוריות שקולות, כל הטובים בשתי ההיסטוריות חייבים להחליט על אותו ערך.

טענה 2.7 יש טרנזיטיביות - אם $H \sim H',\ H \sim H''$, אזי גם $H \sim H''$, והטובים מחליטים על אותו הערך.

הנחות:

1. בשלילה הנחנו כי קיים פרוטוקול P העוצר תמיד ב-t פאזות או פחות.
2. $n > t-2$.

נוכיח את הטענה בכך שנראה שקילות (בטרנזיטיביות) בין היסטוריה שבסופה הערך המוסכם הוא 1, לבין היסטוריה בה הערך המוסכם בסוף הריצה הוא 0 - H_0, ומטענה מעלה נגיע לסתירה.

בהמשך ההוכחה נסתכל על היסטוריות מכל המרחב האפשרי. נתמקד בהיסטוריות שעד פאזה k נפלו לכל היותר k מעבדים. זה יפשט את ההוכחה בהמשך - וזה בסדר, כי מניחים נכונות לגבי כל כל ההיסטוריות, ולכן בפרט גם נכון לגבי ההיסטוריות הספציפיות הללו.

המשחק שנשחק: כל הדרך ניקח סדרה של מעבדים, נפיל ונתקן אותם ע"מ לשמור על שני "טובים" שלא רואים הבדל, ואז "נבלבל" אותם.

אינטואיציה - בציור *איור*... מראים שאם מעבד התקלקל בפאזה אחרונה, אפשר לתקן\לקלקל אותו מבלי לפגוע בערך ההחלטה.

מגיעים להיסטוריה בה q מנותק לחלוטין. להפך, ניתן לצאת מהיסטוריה זו ו"לחזור אחורה" - כל 2 היסטוריות שונות שקולות.

זה נכון **כל עוד יש t מעבדים שניתן לקלקל** - לכן יש להקפיד כי מספר המקולקלים לא יעלה על t.

נרצה לומר שאפשר לעשות זאת תמיד. נוכיח למעשה זוג טענות "יחדיו":

טענת הקלקול

טענה 2.8 בהיסטוריה נתונה, אם עד הפאזה ה-k היו לכל היותר k נפילות, ואין נפילות בפאזות הבאות, ניתן לנתק מעבד בפאזה k ולשמור על ערך ההחלטה.

טענת התיקון

טענה 2.9 אם בהיסטוריה נתונה, עד פאזה k היו לכל היותר k נפילות, ואין נפילות בפאזות הבאות, ניתן לתקן מעבד שנפל בפאזה k מבלי לשנות את ערך ההחלטה.

מה שטענות אלו אומרות - מה שעשינו עד עכשיו בפאזה t, ניתן לעשות בכל פאזה בדרך. נשים לב כי לא יכולנו לעשות זאת ללא ההנחה בשלילה שלנו - ב-t פאזות לא מובטחת לנו פאזה "נקיה" (כלומר פאזה שמעבד לא נפל בה). **הוכחה:** נוכיח את שתי הטענות באינדוקציה יורדת על k, ובמקביל:

בסיס: $k = t$, המקרה באיורים H_1, H_1' , ולכן הטענה נכונה.

צעד: נניח כי שתי הטענות נכונות עבור $k+1$ והלך, ונוכיח עבור k:

תחילה - הקלקול: עד $k-1$ היו $k-1$ נפילות. נבחר מעבד q ונרצה לנתק אותו. נסתכל על כל שכניו - בוחרים שכן p_1. לפני ניתוק q, מנתקים את p_1. לפי הנחת האינדוקציה, קיבלנו היסטוריה בה p_1 מנותק ב-$k+1$: כלומר הקשת בין p_1 ל-q נקטעה, כפי שרצינו.

לאחר ניתוק p_1, בוחרים שכן אחר של q - p_2. אותו גם נרצה לנתק באותו האופן. על כן תחילה נתקן את p_1 - מותר מהנחת האינדוקציה, ואז ננתק את p_2 באותו האופן. נשים לב כי חייבים לנתק ולתקן את השכנים באופן שכזה - אחד אחד, על מנת לעמוד בתנאי הנחת האינדוקציה.

כאשר מתקנים בחזרה את p_1, הוא מנותק מ־q, אבל עדיין שולח את ההודעות שלו, ולכן ערך ההחלטה לא ישתנה (זה נכון לכל פרוטוקול בעולם). ■

מנכונות הטענות לכל k, זה נכון גם למנהיג, ולכן $H_1 \sim H'_1$, בסתירה.

נשים לב ־ לא ניתן להוכיח את הטענה באותו האופן עבור אלגוריתם רנדומי. מה שכן, נראה שאפשר לעצור בתוחלת קבועה של מספר הסבבים ־ אבל זה אומר שבהסתברות כלשהיא אפשר שנעצור לאחר מספר ארוך של סבבים.

ראינו שהעולם לא כל כך פשוט כפי שהוא נראה.

אפשר להסתכל על מודלי שגיאה אחרים שאינם fail-safe ־ המשפט אומר שגם במקרים האלו, צריך לפחות $t+1$ סבבים. יש משפט שאומר שכל אלגוריתם ביזנטי פולינומי עוצר בכל היותר $t+1$ סבבים ־ לא נראה זאת.

היופי ־ למרות שמצאנו חסם תחתון למודל פשוט, החסם תקף גם למודלים אחרים.

2.3 בעיית ההסכמה עם מודל שגיאה ביזנטי

נביט בבעיה במודל שגיאה קיצוני אחר, כמעט מלא, וננסה להגיע לאותה הסכמה.

נניח כי יש לנו היכולת **לחתום** על הודעות, כך שרעים לא יכולים לזייף חתימה של טובים, וכך שלאחר חתימה, כל אחד יכול לראות מי חתם על ההודעה.

תזכורת: מודל שגיאות ביזנטי ־ מעבד "רע" יכול לעשות מה שהוא רוצה, במקרה הזה במטרה למנועה הסכמה.

ובאופן יותר פורמלי:

הנחה: קיום חתימות דיגיטליות, מודל שגיאות ביזנטי, מערכת סינכוני ורשת מלאה.

השאלה: האם משהו בכוח של ההוכחה הקודמת יכול לעזור לנו במקרה הזה, בהנתן הכלי החדש (חתימות דיגיטליות)?

רע יכול לשלוח הודעה, אך לא יכול לטעון שקיבל אותה מטוב אם זה לא נכון ־ אנו מניחים שאנו יודעים מי כל אחד, מי המנהיג, וכו'.

הכח של החתימה ־ לא ניתן לזייף את החתימה של המנהיג, אם המנהיג טוב. אם המנהיג רע, אפשר להחליף הודעות כאלה, שכן רע יכול לחתום בשם רעים אחרים.

הקושי ־ רע יכול לחתום על ערכים שונים, קרי יהיו מספר ערכים שונים במערכת. אזי לטובים יהיו בפאזה האחרונה מספר ערכים ־ איך יחליטו ביניהם?

אבחנה 1: אם יש הוכחה שהמנהיג רע ־ כלומר אם יש שתי הודעות שונות חתומות על ידו, אז אם כל הטובים רואים את ההודעות הללו, יש להסכים על ערך $\perp$. אם זה קרה רק בפאזה האחרונה, יש לטפל בזה.

המקרה הקל הוא המקרה בו המנהיג טוב ־ כולם יקבלו הודעה חתומה ממנו, ולא תתקבל הודעה חתומה אחרת שלו.

דבר ראשון שצריך למנוע ־ בשניה האחרונה מישהו יציג לנו הודעה ש"תקלקל" הכל, ואז לא נוכל לתחום את מספר הפאזות.

איך נקשור בין ההודעה לחתימת זמן?

נספור את **מספר החתימות**, ולפי זה נוכל לדעת ־ לכל הודעה יהיו לכל היותר t חתימות של רעים, ולכן הודעה "רעה" לא תוכל להגיע למצב בה יש לה $t+1$ חתימות שונות.

פרוטוקול ראשוני

- **פאזה 1**: המנהיג חותם ושולח.

- **פאזה** i: אם קיבלת בפאזה ה־$i-1$ הודעה חתומה עם $i-1$ חתימות שונות - חתום ושלח.
- **בסוף הפאזה ה־**$t+1$: ?

שיעורי הבית:

1. השלם את הפרוטוקול מעלה לפרוטוקול תקין.
2. נביט בבעיית ההסכמה במודל שגיאות fail-safe, ונניח כי גם יש מטבע משותף רנדומי - לא יודעים את ערכו לפני שמטילים, וכאשר מטילים אותו, כולם מקבלים את אותו הערך. בכל פאזה מותר להטיל אותו. כתבו פרוטוקול עם תוחלת מספר סיבובים קבועה. אתגר: כתבו פרוטוקול שתמיד יעצור ב־$t+c$ פאזות עבור c קבוע כלשהוא.

3/12/2012
10/12/2012 יושלם בהמשך...

לגבי התרגילים שקיבלנו - עדיין המרצה לא עבר על אף תרגיל. דני יתן את הפתרונות שהוא יכול לחשוב עליהם (כי הוא מניח שזה מעניין).
היום נוכיח את הגבול התחתון של שליש. אותם רעיונות בן־אור הוכיח לאחרונה עבור קוונטי. אח"כ נעשה משהו שהמרצה לא עשה בקורסים הקודמים - התמודדות עם שילוב של ביזנטי במערכת סינכרונית.
תחילה נתייחס במהרה לפתרונות של תרגילי הבית:

2.3.1 תרגיל בית 1

לשלוח רק שני ערכים - ברגע שקיבלנו שנים, אנו יודעים שהם שגויים, לכן מספיק לשלוח רק שניים. אם קיבלנו אחד - לקבל.

2.3.2 תרגיל בית 2

יש הרבה שימושים למטבע, ככל שמודל השגיאה מסובך, צריך לעשות שימוש יותר מסובך.
במודל ה־$Fail-Stop$ אפשר להשתמש במטבע בצורה קצת שונה מצורת החשיבה המקובלת.
הבעיה: אם שולחים גם אפס וגם אחד באותה פאזה. הדרך למנוע: מכיוון ובמודל זה אין "שקרנים", השימוש המתוחכם הוא:
אם המטבע שווה לערכך,שלח את ערכך. אם קיבלת ערך ממישהו - עדכן לערך הזה. אם לא קיבלת - חכה. בסוף $t+1$ החלט על ערכך - בסוף הפאזה הזו יש פאזה אחת בה אין נפילות, וכפי שראינו זה מספיק לנו (ברגע שהגענו לשלב שכזה, אפשר להסכים).
בפעם הראשונה שהמטבע זהה למטבע של אחד הטובים - מחליטים על זה. אם כל הטובים מתחילים על אותו ערך - בפעם הראשונה שזה קורה כולם שולחים, מחליטים ומסיימים. אחרת, בפאזה שלאחר מכן לכולם יש את אותו הערך, וכולם מחליטים. תוחלת מספר הסיבובים - 2 או 4, תלוי איך "עוטפים" את זה.
זהו המקרה הכי קיצוני, אבל זהו <u>אינו המקרה המקובל</u>.
טוב, האמת היא שכאן דיברנו על בעיית קונסנזוס (ללא מנהיג), אבל אפשר לעשות אדפטציה קלה למקרה שקיבלנו בשיעורי הבית.

2.3.3 לפתרון לבעיה הביזנטית במערכת סינכרונית אי אפשר להתגבר על יותר משליש רעים

בפרוטוקול בשבוע שעבר הסתמכנו על כך ש־$n \geq 3t+1$. נשאלת השאלה, האם זהו אכן גבול.

במהלך השנים לא היתה הוכחה נקיה לעניין זה. לאחר מספר שנים של נסיונות, ננסי לינץ', מייקל פישר ומייקל מרי מצאו את הדרך הנקיה ביותר להוכיח זאת.

נביט במקרה בו יש רק שלושה:

איור

נביט במקרה בו A שולח ל־B, C, והם מנסים להחליט מה נשלח.

אם B אומר ששמע 1, ו־C אמר ששמע 0. ישנן שלוש אפשרויות:

- A רימה.
- B רימה.
- C רימה.

איך נקבע מי שיקר? מספיק שיש שקרן אחד, ואז יש סימטריה שאי אפשר להתגבר עליה. אולי יכול להיות שאפשר להתגבר על הבעיה בעזרת פקוטוקול מסויים. יש צורך לפרמל את האינטואיציה הזו לגבי כל פורטוקול שבעולם.

הדרך המתוחכמת שהשלושה מצאו היא הבאה:

רוצים להוכיח שאין פרוטוקול שבשליש רעים מגיע להסכמה.

נניח בשלילה שיש פרוטוקול כזה. נתחיל ב־$n = 3$. נניח שיש דרך פלאית בה אם מזינים ערכי התחלה כלשהם (בינריים לשם פשטות) לשלושה ואם נשתלט על אחד מהם, אז שני האחרים תמיד ידיעו להסכמה (קונצנזוס - ללא מנהיג).

נזמין שני העתקים של המכונה הפלאית הזו. במקום לנסות לאמר מה המכונה עושה ואיך מרמים אותה, נעשה את הדבר הבא:

ננתק את הקו מ־A ל־C, A יקבל ההודעות מ־C', ו־C יקבל ההודעות מ־A'. בחיבור זה מגדילים את הסנריו עבור הרע:

נניח ש־A, B הם הטובים. אזי הם שני מעבדים שכנים במקורית, וכל היתרה היא מה ש־C יכול "לשחק" איתה. מותר לו לעשות את זה. אזי, אם הטובים מתחילים עם אותו ערך, הם צריכים להסכים עליו. היריב מדמה כאילו A', B' אומרים מה ש־A, B אמרו, ואז "משחק" עליהם. אין עם זה בעיה, כי אין כאן חתימות (זהו מודל ביזנטי בסיסי). זהו הכלי של היריב, והדרך לפרמל את האסטרטגיה של הרע. כלומר:

נטען כי מבחינת נקודת ראותם של כל המעבדים במשושה, הם יושבים במשולש (כלומר "עולם" במשושה הוא עולם אפשרי במשולש - אם אחד מהשישה היה יכול להבדיל אם הוא במשולש או במשושה, כל ההוכחה היתה נופלת), כי השכנים שלהם הם השכנים שקיימים במשולש: C שולח ל־B ול־A (למעשה ל־A', אבל זה לא משנה ל־C), מקבל מ־A, B (למעשה מ־A'), A מקבל מ־B ו־C (למעשה מ־C') וכו', והם מריצים את הקוד הפלאי.

המשושה מגדיר כל זוג קודקודים שהם טובים, וכל סנריו הוא סנריו שהיריב יכול להמציא לכל פרוטוקול שבעולם. אפשר לדחוף גם את המודל הזה למודל הסתברותי (צריך להגיד במקרה זה מה הסיכוי ל, אבל נתעלם מזה ונשאר עם מודל דטרמיניסטי). כל קודקוד מעביר לשכניו את ערכו.

אם שלושת המקוריים התחילו עם 1, נזין למדומים את ערך ההתחלה 0 (נזין תמיד את ה"משלים"). המדומים אם כך חייבים להסכים על ערכם (כי הם יהיו שם שני טובים שהתחילו עם אותו ערך התחלה). אם B "הרע", A ו־C צריכים להסכים. מכיוון ו־C מחוייב להסכים 0 כי הוא אינו יודע מיהו השקרן, גם הוא מחויב להסכים אפס. אבל A, B התחילו עם אחד, ולכן צריכים להסכים על אחד (וכך לכל זוג), בסתירה לכך ש־A, C היו חייבים להסכים על 0. על כן, המשולש לא יכול להגיע להסכמה, ללא תלות בפרוטוקול.

ההוכחה הזו מאוד חזקה, כי אין כאן תלות בכלל בפרוטוקול - מפתה מאוד לחשוב שיש איזה שיפור, אבל מכיוון ולא היה כאן שימוש בשום תכונה של פרוטוקול, אין זה אפשרי.

נכליל את ההוכחה ל־n כלשהוא, ונניח לשם פשטות כי $n = 3t$:
נחלק את n לשלוש קבוצות, שגודל כל קבוצה הוא t, נסמנן ב־$\tilde{A}, \tilde{B}, \tilde{C}$ (אם $n < 3t$, בכל קבוצה יהיו לכל היותר t מעבדים). נניח שהמכונה הפלאית יודעת לעשות להגיע להסכמה ל־n גדול כלשהוא. נבצע רדוקציה למשולש:
יש הודעות בתוך כל קבוצה, ויש הודעות מקבוצה לקבוצה.נניח כי הגרף מלא. הפרוטוקול הוא כזה שכל אחד שולח את ההודעה שלו לכולם, ובשלב מסויים מגיעים להסכמה. אם יש הסכמה במקורי, יש הסכמה גם אם נותנים לכל קבוצה ערך התחלה - נניח $\tilde{B}$ מתחילים עם 0, $\tilde{C}$ מתחיל עם 1.
הפרוטוקול למשולש יהיה - $\tilde{A}, \tilde{B}, \tilde{C}$ ישכפלו את עצמם t פעמים, ויריץ את הפרוטוקול עבור n קודקודים. זהו פרוטוקול לגיטימי לחלוטין. זהו פרוטוקול שלמשולש היה צריך להסכים גם כן, אם הסכים על $n = 3t$ (כאילו "מפילים" את אחת מהקבוצות, ולא חלק פה וחלק שם - אם אחת מהקבוצות היתה מגודל גדול מ־t, לא יכולנו לבצע זאת). קיבלנו מיפוי מ־$n = 3t$ למשולש, וזו סתירה, שכן הוכחנו שלמשולש לא קיים פרוטוקול כזה, ועל כן לא קיים פרוטוקול שכזה עבור n כללי.
בכלי זה - סימולציה וירטואלית, משתמשים בהוכחות רבות.

2.4 מודל $Self - Stabilization$

המודל הזה של שגיאות ברות חלוף היה - הקוד עצמו צרוב אצל הקודקודים, לכן הוא לא משתנה. כל מה שמשתנה בתוכנה יכול להשתנות. אז האסטרטגיה של הרעים יכולה להיות מניעת התכנסות באופן כללי (לא רק מניעת הסכמה על אותו ערך).
כלומר, כאן לא לוקחים כמובן מאליו את הידיעה של המנהיג, את הידיעה "מי אמור לשלוח מה", וכו'. אין מספרי פאזות (למרות שהמודל סינכרוני - כולם יודעים שמתחילה פאזה, אבל זה לא אומר כלום - אולי אני חושב שאני בפאזה מסויימת, והאחרים חושבים שהם בפאזה אחרת). אלו חלק מהקשיים שמתמודדים איתם כשמנסים לשלב את המודל עם שגיאות ביזנטיות. עוד בעיה לדוגמא: טוב שולח הודעה, חלק מהטובים חושבים שקיבלו בפאזה מסויימת, אחרים בפאזה אחרת. עדיין נרצה לוודא שהמערכת מתייצבת.
נחליט שהבעיה היא החלטה על מספר הפאזה. למה? אם בתנאים אלו היה פרוטוקול שידע לפעול כשיודעים מהו מספר הפאזה, היינו יכולים להריץ לפני כן פרוטוקול שיגרום לכך שנדע מהו מספר הפאזה, וכך נוכל לפתור את הבעיה. לכן, עלינו קודם כל לדעת איך לפתור את הבעיה הבסיסית ביותר - קבלת מונה פאזות.
המודל שנתמודד איתו הינו פשוט:

המודל נניח שיודעים מתי מתחילה פאזה, והפאזות מרווחות כך שאם שולחים הודעות בפאזה, אז כולם מקבלים באותה פאזה החל מהרגע שבו המערכת מתייצבת. נחשוב על זה ככה: עד רגע היצוב כולם רעים, מאותו רגע יש מספר רעים מסויים. כמו כן, לאחר ייצוב המערכת, היא יודעת "לנקות את עצמה" בכל פאזה.

המטרה החלטה על מספר הפאזה.

זוהי לא סתם בעייה תיאורתית - זוהי בעיה אמיתית, שכן פרוטוקול תמיד רץ, ולא באמת יודעים מתי הפרוטוקול התחיל לרוץ. למשל, מחשב שעשה $hibernation$ והתעורר מחדש לא יודע שישן, רק יודע שעבר זמן.
מה שמעניין אם כך, למצוא מה אפשר להגיד בהנחת מודל השגיאה הכי חזק, ולראות כמה אפשר לדחוף את המעטפת. עד לפני 10 שנים בערך לא היה ברור שאפשר לעשות זאת (האלגוריתמים עד אז השתמשו בהטלות מטבע, וההתכנסות היתה מאוד, מאוד, מאוד

איטית). בשנים האחרונות הופיעו כמה תובנות שהביאו להתקדמות, באחת מהן נדון היום. לכן נתמקד במודל הפשוט אותו ציינו מעלה.

2.4.1 הסכמה ביזנטית

נרצה להגיע הסכמה ביזנטית כמו שראינו (לאו דווקא עם חסם על t כפי שראינו, אולי ניקח חסם יותר נוח לשם התחלה): יכולנו להריץ את הפרוטוקול שראינו בשבוע שעבר, אבל אם אי אפשר לדעת באיזו פאזה אנחנו, זה לא יעבוד. לו הפרוטוקול היה מתחיל אצל כולם באותה פאזה במודל שלנו, לאחר Δ פאזות היינו מקבלים אאוטפוט קונסיסטנטי יחסית למצב לפני Δ פאזות, שכן הפרוטוקול היה עובד.

כיצד נוכל לפתור את הבעיה, בהנתן חסם מסויים על הרעים?

בכל פאזה צריך לבצע GC. אם כך נתחיל בשאלה - איך עושים GC בסביבה שכזו? אם הוא לוקח בדיוק פאזה, אין בעיה, הקושי הוא כאשר תהליכים לוקחים יותר מפאזה אחת.

נחלק את הזיכרון של ה־GC של מעבד לשלושה: אחד שהתחיל בפאזה הזו, שני שהתחיל לפני פאזה, שלישי שהתחיל לפני שתי פאזות, כלומר מתחילים הרצה בכל פאזה. על כן, לאחר סיום גל השגיאות נגיע להסכמה בכל העותקים שהתחילו החל מאותו הרגע. נשים לב:

- אנחנו לא יודעים מתי גל השגיאות הסתיים.
- ערך ההסכמה תלוי בערכי ההתחלה, לכן כל עותק שכזה שהתחיל לאחר סיום גל השגיעות יתן תוצאה נכונה, אבל כל תוצאה שכזו תלויה בערכי ההתחלה, לכן לא בהכרח נקבל את אותה התוצאה עבור אותו העותק.

האם אפשר לעשות זאת לפרוטוקול הכללי? כן! צריך פשוט Δ עותקים - הפאזה ה־i תריץ את העותק ה־i[3]. קיבלנו מעין $pipeline$, כאשר בכל פאזה Δ הודעות. בהתחלה מקבלים זבל, כאשר לאחר שלב השגיאות כל עותק חדש שיתחיל לאחר Δ פאזות יתן לנו $output$ נכון - הסכמה. עם זאת, כמו עבור GC, ההסכמה בכל עותק שהתחיל לאחר ההתייצבות לא תהיה בהכרח אותה התוצאה, שכן התוצאה תלויה בערכי ההתחלה. זהו כלי בלבד, ונראה איך משתמשים בו בהמשך.

תובנות - אילו כלים עומדים לרשותנו

- אם יש "בלאגן", אפשר לאפס. לראות אם יש בלאגן אפשר לעשות על סמך ההסכמה, או על סמך מה שאני שולח כרגע. אז קיבלנו בעצם שני כלים - אם מעבד שולח בכל פאזה מהו חושב שמספר הפאזה הנוכחי, אפשר להשתמש גם בזה (הסכמה חלקית) - אם כולם מסכימים למשל על מספר הפאזה, אפשר להמשיך עם זה.
- יש מונה פאזות c_p, שצריך להגיע להסכמה עליו (**לרב** הטובים - רב זה תלוי בפרוטוקול שנבחר), והוא צריך לגדול ב־1 בכל פאזה "כל עוד הכל בסדר". מונה שכזה צריך מודולו מסויים (כדי לא להגיע לסיום), ולשם פשוט נניח כי המודולו קטן מאיזשהוא כפילות של Δ.
- יש תוצאת ההסכמה שהסתיימה עתה v_p. אפשר להשוות למשל בין v_p ל־$\overline{v}_p$ - ערך ההסכמה הקודמת, או בפאזה שהיתה לפני Δ. גם v_p הוא באותו מודולו כמו מונה הפאזות.

[3] יש כאן כביכול בעיה של סיבוכיות, אבל קודם המטרה לפתור הבעיה, אח"כ לשפר את הסיבוכיות.

- אפשר לשמור רשימת רעים, אבל כל אחד יכול להכניס לשם טובים. לכן פרוטוקול שמשתמש ברשימה שכזו צריך להתרענן כל פאזה. לכן משתמשים בזיהוי מקומי של "רעים" (למרות שב־GC יש זיהוי מסויים של רעים, הרשימה לא נשמרת הלאה, וצריך לראות אם אפשר לגרור אותו הלאה, ואם כך, כיצד).

- בכל פאזה צריך להחליט מהו האינפוט שיהיה לפרוטוקול ההסכמה של הפאזה הבאה. אפשר להחליט על תלות בין האינפוט הזה לתוצאת ההרצה של ההסכמה שהסתיימה עכשיו. למשל, הגדלה באחד בכל פאזה, או השוואה עם "ערכי" לפני Δ, ובכל שלב של חוסר קונסיסטנטיות אפשר לאפס (יש לשים לב שהרעים לא גורמים לנו לאפס כל הזמן).

- בכל פאזה בנוסף אפשר לשלוח הודעות נוספות v־אים מלבד ההסכמות: לשלוח מה הערך שלי, ולקבל מהאחרים את ערכם. זהו לא פרוטוקול הסכמה, אך זהו עוד כלי שיכול לעזור לנו לבדוק את הקונסיסטנטיות של המצב.

בעקרון אפשר לקבל את c_p, v_p כך שיהיו מונוטונים עולים במדולו. הבעיה היא שצריך איזשהוא כלי נוסף, כי אי אפשר לסמוך על מה שהיה ב־Δ. כלי נוסף למשל שציינו הוא המונה הנוכחי אצל האחרים.

נניח ששולחים גם את c_p, ואז כל מעבד אוסף את c_p אצל כולם. אפשר למשל אם יש רע אחד לפחות לאפס, או לקחת את הערך שחוזר רב הפעמים להיות המונה, ואם אין ערך שהרב מחזיקים בו, לקחת אפס. יתכן וזה ישנה את היחס בין טובים לרעים, אבל על זה נדבר בהמשך. היינו רוצים להגיע למצב בו אם אנחנו רואים רוב לערך אחד, אחרים לא יראו רוב לערך אחר.

פרוטוקול ראשוני

1. שלח את Δ ערכי ההסכמות.
2. שלח את c_p.
3. הרץ את כל Δ העותקים.

בסוף שלב זה, יש את v_p, ואת הוקטור C של כל מה שקיבלנו מכל ה־n־ית המעבדים.

4. הגדר c_p להיות ערך הנמצא לפחות $\left\lceil \frac{n+1}{2} \right\rceil$ פעמים ב־C, **אחרת** $c_p = 0$.
5. אם $v_p = \overline{v}_p + 1$ או $v_p = 0$, אז $c_p := c_p + 1$, **אחרת** $c_p = 0$.
6. $\overline{v}_p := v_p$.
7. ערך ההתחלה של ההסכמה החדשה יהיה c_p (תתחיל בפאזה הבאה).

מה אפשר להגיד על הסכמה הראשונית הזו?

- אם מתחילים בעולם נקי יחדיו, האם הפרוטוקול עובד? ב־Δ הפאזות הראשונות מקבלים אפסים כל הזמן. לאחריהן אם כל הטובים שולחים אותו ערך, ואם היחס בין הטובים לרעים לפחות שליש, יש רוב ולכן c_p יגדל. מכיוון והמונוטוניות תמשך, גם v_p יגדל בהתאם, והכל טוב.

- האם לאחר התייצבות, יתכן ושני קודקודים יאפסו את שני המשתנים בו זמנית? חלק 5 תלוי רק בהסכמות. אם מקבלים סדרת הסכמות טובה, אז בסופו של דבר כולם יתחילו עם $v_p, \overline{v}_p$, כלומר התנאי בשלב הזה מבטיח כי כל הטובים בשלב כלשהוא יעשו אחת משתי הפעולות, כלומר זהו קו מפריד. אם פעם אחת כולם מתאפסים, אפילו פעם אחת - סיימנו, שכן זהו הריסט הגלובלי - כולם את הפאזה הבאה יתחילו אם $c_p = 0$, הוא יהיה קונסיסטנטי אצל כולם ולכן בפאזה הבאה c_p יתייצב ולא ישתנה בשלב 4 לאף אחד, ומכיוון והסדרה תמשך להיות טובה כל הטובים יבצעו אחת משתי הפעולות, ולכן לאחר זמן מסויים כולם הפרוטוקול יסתיים.

על כן, המשימה היא לגרום לכל הטובים "לדרוך" פעם אחת ב־5 כך שכולם יאפסו את c_p - כלומר למנוע מ־v_p לגדול מונוטונית. מהם התנאים הדרושים לכך? כרגע אין שום קשר בין ערכי c_p לבין v_p, וזהו השלב שחסר לנו.

הבעיה היא שכל הזמן חלק יכולים לראות ערך אחד, אחרים אחר, ואז v_p נשאר מונוטוני. אפשר למשל, לחזק את את פרוטוקול ה־$Agreement$ לרב מספיק טוב, אחרת נאפס:

- קדם הסכמה: שלח $input$ (c_p) לכולם. אם קיבלת $n - t$ ערכים זהים, זה הקלט להסכמה, אחרת $\perp$.

נשים לב כי כאן מגדילים את Δ ל־$\Delta + 1$. אזי, אם הרבה פעמים במסלול 5 הרבה מראים 0 והרבה 1, יהיה ריסט להסכמה v_p (אם מצליחים לווסת את היחס בין הטובים לרעים). מדוע התנאי הזה מספיק?

- כאמור, אם בשלב חמישי פעם אחת כולם איפסו את c_p, נגיע בסופו של דבר (בזמן סופי) למונה זהה אצל כל הטובים.
- **אם יש $t + 1$ עם ערך אחד ו־$t + 1$ עם ערך אחר**, ההסכמה v_p תעשה ריסט, כלומר הטובים יתחילו עם $\perp$, לכן האוטפוט שלו יהיה $\perp$ וכולם יסכימו על אותו ערך עוד Δ פאזות. מספיק $\frac{1}{4}$ רעים למקרה זה (שליש לא טוב מספיק).
- **אם יש לכל היותר t טובים עם ערך שונה מכל השאר**, ההסכמה תמשיך להתגלגל כשורה. שלב 4 בפאזה הבאה יאפס את c_p, ונגיע לתוצאה הרצויה.

זה לוקח בערך $3D$ סבבים, כאשר ערך המודולו הוא D.

נשים לב כי לאחר הפעם הראשונה שנתחתנו בשלב 5 על אפס, נקבל ב־Δ הבאות סדרות מונוטוניות או אפסים (בהתאם לערכים ההתחלתיים), אבל מובטח שכולם מכניסים את 0 או את c_p המונוטוני ביחס לקודם, שכן 4 כבר לא מקלקל את c_p. לאחר ה־Δ נקבל סדרות מונוטוניות עם אפסים ביניהם, לכן המונה c_p ממשיך להתקדם מונוטונית, ונסיים.

2.5 בעיית ההסכמה במערכת אסינכרונית

2.5.1 משפט FLP

17/12/2012

היום נחזור לעולם האסינכרוני, ונראה מה ניתן לעשות בו בהנתן שגיאות אפשריות.

היכן הקושי בעולם האסינכרוני? אין לנו סיום של פאזה שיצביע על כך שלמשל השולח שגוי. לכן לא ניתן להבדיל בין פעולה שלוקח יותר זמן לבין שגיאה.

במשך שנים היו ידעו שלא ניתן להגיע להסכמה במערכת כזו. לקח זמן עד שהגיעו להוכחה בצורה נקיה - ואז ראו שהוכחה זו נוגעת למודלים רבים ושונים. נניח מערכת אמינה. יש מודלים נוספים בהם timeout לא מספיק אמין, ואז הבעיה הזו תקפה. לא נפתח את כל משמעות המשפט במסגרת הקורס, אך צריך להיות מודעים לעובדה שאם יש

$timeout$ במקום מסוים, אין זה אומר שיש במקום אחר. אם יהיה מודל בו אם מישהו שולח הודעה היא מגיעה לכולם, ואם מישהו אחר שולח ההודעות יגיעו לפי הסדר, אין צורך ב־$timeout$, כלומר אין תמיד צורך בכך בעזרת חיזוק המודל. המשפט עצמו בעקרון הוא משפט מאוד בסיסי, והוכחתו כאמור היא מאוד נקיה.

משפט 2.10 אין פרוטוקול המתגבר אפילו על שגיאה בודדת במערכת אסינכרונית.

כלומר אפילו לא במודל fail-stop.

כדי להוכיח משפט כזה, כפי שראינו בקורס לפחות פעמים, צריך דרך לדבר על כל הפרוטוקולים האפשריים. בסביבה האסיכרונית, הכלים שראינו לא עוזרים לנו. כאן צריך למדל את הפעולות השונות, את המצבים השונים בהם מערכת יכולה להיות, ולהסביר "מה זה" להגיע להסכמה במערכת כזו.

נתחיל באחרון ־ המערכת מתחילה במצב מסוים, כל מעבד הוא $state-machine$, יש לו ערך $input$ וערך $output$ שבשלב מסויים הוא מדליק ־ אחרי הדלקת הערך המעבד יכול להמשיך לעבוד, אך הדלקתו פירושו שהמעבד החליט.

אנו נוכיח לשם פשטות למודל הבינארי לבעיית הקונצנזוס ־ ולכן הסכמה היא כי הערך $output$ זהה, ואם כולם התחילו עם ערך $input$ זהה, זהו הערך איתו כולם צריכים לסיים.

מידול המערכת אם יש שני מעבדים שבו זמנית כל אחד קורא הודעה שמגיעה אליו ומחליט לשלוח קבוצת הודעות, אם היינו מבצעים את הפעולות במקביל או אחת אחרי השניה ־ אין לכך חשיבות, כי ההודעות שנקראות במקביל הן הודעות שנשלחו קודם לקריאה, לכל אחד יש את ההודעה שמחכה לו, ולכן לסדר הקריאה אין חשיבות. אם אחד שולח הודעה לפני השני אין לכך חשיבות, כי במערכת אסינכרונית כאמור אין לכך חשיבות. אם כך יש כאן דרגת חופש, בה נשתמש בהמשך.

איך ניתן לבטא צעד ב־$state\ machine$ של מעבד? מערכת אמינה אומרת שכל הודעה שנשלחה תגיע ליעדה.

מאורע\צעד $e=(p,m)$: נסיון של המעבד p לקרוא הודעה מהרשת m, כאשר $m\in M\cup\{\phi\}$ (יתכן והיא ריקה ־ כלומר אין הודעות), ביצוע חישוב, ושליחת הודעה. נסמן ב־C_i המצב הפנימי של ה־$state-machine$ של מעבד p_i. אזי ביצוע פעולה\מאורע e היא $e\left(C_i\right)\to C_i'$.

בהנתן מערכת עם n מעבדים, כל מעבד מתחיל ממצב התחלתי מסויים $C_1^0,...,C_n^0$, וכל הזמן המעבדים באופן אסינכרוני מבצעים צעדים, ובשלב מסויים הם מדליקים את ערך ה־$output$ ־ O_i.

בכל רגע נתון אם כך ניתן להגדיר את מצב המערכת:

$$C=\times_{i=1}^{n}C_i\cup\tilde{M}$$

כאשר $\tilde{M}$ היא אוסף ההודעות שנמצאות כרגע ברשת (כל הודעה נמצאת או אצל המעבד אליו היא מיועדת, או שעדיין לא הגיעה ליעדה).

בצורה כזו יש דרך למדל את מצב המערכת, ואיך מצב המערכת משתנה בכל צעד.

טעויות אפשריות נשים לב כי במודל זה ניתן לומר מתי מאורע הוא חוקי ־ למשל, מאורע לא חוקי הוא מאורע של הבאה של הודעה למעבד שעדיין לא נשלחה עליו. זהו למעשה המאורע הלא חוקי היחיד במצב זה:

לגבי המצבים ־ כביכול אפשרי להגיע למצב לא חוקי, אך מכיוון ומתחילים ממצב התחלתי וממשיכים לפי פרוטוקול, לא ניתן להגיע למצב לא חוקי

לגבי הפעולות - אפשר להפעיל את המאורע מתי שרוצים, והטיפול עצמו תלוי בהודעה.
אפשר להסתכל לא רק על מאורע יחיד, אלא על סדרת מאורעות $\sigma = e_1, ..., e_n$ שאפשר להפעיל החל ממצב מסויים C_σ.

למת המעויין

למה 2.11 יהי C מצב מערכת, ותהיינה σ_1, σ_2 סדרות של פעולות.
אם C_{σ_1} ו-C_{σ_2} תקפות, ו-σ_1 ו-σ_2 זרות במעבדים, אז:

$$C_{\sigma_1\sigma_2} \equiv C_{\sigma_2\sigma_1}$$

איור

מדוע הלמה נכונה?
כל ההודעות שנצרכו ע"י מעבדים ב-σ_1 הן או הודעות שהיו זמינות כבר ב-C, או שנוצרות במהלך σ_1. כנ"ל לגבי σ_2. לכן אחרי ריצה של $\sigma_1\sigma_2$ ולאחר ריצה של $\sigma_2\sigma_1$, אותה קבוצת הודעות יצאה מ"שק" ההודעות, ואותה קבוצה נצטרכה בסופו של דבר (יתכן ומעבדים מ-σ_1 שלחו הודעות שנצטרכות ב-σ_2, אבל אלו רק הודעות שכבר היו זמינות ב-C, ולהפך). כלומר ה-n-איה של המעבדים נמצאת באותו המצב סופי.
למה זו מגדירה את דרגת החופש שלנו.
נוכיח למה ראשונית נוספת, ואחר מכן נעבור למשפט הראשי.

למה נוספת כדי לנסח את הלמה, עלינו להגדיר על המצב "להגיע להחלטה". יש אינסוף סדרות אפשריות של פעולות המתחילות מהמצב ההתחלתי. סדרה מחליטה היא סדרה בה בשלב מסויים מעבד מסויים מחליט - היא נקראת כך מכיוון ואחרי שמעבד אחד החליט, אז השאר יחליטו על אותו הערך.
מצב מערכת הוא דו ערכי, אם יוצאת ממנו לפחות סדרה מחליטה אחת בה יש מעבד החליט 0 (ולכן כל השאר גם יחליטו 0), וסדרה מחליטה אחרת בה יש מעבד המחליט 1 (ולכן השאר גם יחליטו 1).
מצב המערכת הוא חד ערכי, אם כל הסדרות המחליטות היוצאות ממנו מגיעות לאותו ערך החלטה (למרות שיכול להיות שבמצב זה עדיין אף אחד לא החליט עדיין).

למה 2.12 יש מצב התחלתי דו ערכי.

כלומר, לא משנה מה הפרוטוקול, בהנתן שאנחנו יכולים לקבוע את ערכי ההתחלה, קיים מצב התחלתי דו ערכי.
נשים לב כי אין זה ברור מעליו: מכיוון ואין רנדומליות, המערכת כביכול דטרמיניסטית. כמו כן, אם אין נפילות, אין מצב התחלתי דו ערכי - ערכי ההתחלה קובעים באופן יחיד את ערך ההחלטה. **הוכחה:** נסמן ב-C_0 את המצב בו כולם התחילו עם אפס - זהו מצב חד ערכי עם אפס. בצורה דומה, C_1 הוא המצב בו כולם התחילו עם אחד - זהו גם מצב חד ערכי, אולם עם 1. נסתכל על כל מצבי ההתחלה האפשריים - כולם בין שתי האופציות האלה -

$$C^0 \;\; (0,0,...,1) \;\; (0,0,....,1,1) \cdots (0,1,1,...,1) \;\; C^1$$

אם המצב $(0,0,...,1)$ הוא חד ערכי (הוא עם ערך 0), נמשיך לבא. כנ"ל מהכיוון השני - אם המצב $(0,1,...,1)$ הוא חד ערכי עם ערך 1, נמשיך הלאה. אם בדרך מצאנו מצב דו - ערכי, סיימנו.

אחרת, לפחות פעם אחת ערך ההחלטה חייב להתחלף מ־1 ל־0 או ההפך - כלומר יש שני מצבי התחלה שכנים C', C'', כלומר הנבדלים זה מזה בדיוק בערך יחיד המקבל מעבד p_k, כל היתר מקבלים אותו ערכי $input$, כאשר מצב אחד מהם הוא 1 ערכי, והשני 0 ערכי. נפיל את p_k ונריץ את כל הריצות האפשריות משני מצבים אלו. מכיוון והזו ההבדל היחיד בין שני מצבי ההתחלה, והפלנו את p_k, בשני המצבים האלה חייבים להגיע להסכמה על אותו הערך - ולכן לפחות אחד מהם חייב להיות דו ערכי.

הערה 2.13 יתכן ולאחר ה"הפלה" הערך השתנה מהערך שהיה קודם לכן, אבל עדיין ערך ההסכמה בין שני המצבים חייב להיות זהה.

■

הלמה המרכזית

למה 2.14 יהי C מצב דו ערכי, ו־$e = (p, m)$ מאורע תקף ב־C (כלומר שניתן להפעיל אותו ב־C). אזי יש סדרה σ (יכולה להיות ריקה) כך ש $e\left(C_\sigma\right)$ הוא מצב דו ערכי.

דהיינו, בכל פעם שהפרוטוקול עומד להחליט, ניתן לחסום אותו.
לפני שנוכיח אותה, נראה מהו הכח שלה - נוכיח את המשפט המלא בהנחה שהלמה מתקיימת:

הוכחת המשפט הוכחה: נתחיל ממצב התחלתי דו ערכי (קיים לפי למה קודמת), נסדר את המעבדים בסדר מסויים וניתן לכולם להתחיל לרוץ. לפני שנשלח הודעה, בודקים אם העברת ההודעה תגרום למעבר למצב חד ערכי. אם לא, נמשיך ונשלח. אחרת, "נאחסן" את ההודעה, נריץ את σ, הקיימת לפי הלמה המרכזית, ונעביר לאחר מכן את ההודעה שאיחסנו. מכיוון וכאמור אנחנו במערכת אסינכרונית, אין בעיה עם התהליך הזה, שכן סדר קבלת ההודעות לא חשוב.
כך יוצרים ריצה אינסופית בה כולם מקבלים את כל ההודעות, אך לא מגיעים למצב החלטה (ואפילו לא מפילים אף אחד מהמעבדים). על כן, לא יתכן פרוטוקול הסכמה המתגבר אפילו על שגיאה בודדת במערכת אסינכרונית, שכן התנאי של זמן סופי לא מתקיים. ■

נשים לב כי מספיק ה"פחד" שאחד המעבדים יפול, שכן לא היינו צריכים להפיל אף מעבד בתהליך אותו תיארנו בהוכחה מעלה.

הוכחת הלמה המרכזית הוכחה: נתחיל ממצב C שהוא דו ערכי, ומאורע e. אם מביא למצב דו ערכי, סיימנו, על כן נניח כי ביצועו לאחר C מביאה למצב חד ערכי, ובה"כ למצב בו ערך ההחלטה הוא 1. מכיוון ו־C דו ערכי, קיימת ריצה אחרת המביאה להסכמה על ערך 0.
תחילה נניח כי בריצה זו לא משתתף e. אזי אם נריץ את e בסוף הסדרה, גם אחריו ערך ההחלטה יהיה 0. נבדוק "אחורה" - נעלה במעלה עץ הריצה, וננסה במקום צעד להפעיל את e. אם לאורך הריצה הזו במקום מסויים נשארנו במצב דו ערכי, סיימנו. אחרת, קיים מצב בו הפעלה של e לפני e' הביאה למצב חד ערכי עם ערך החלטה 1, ומצב בו הפעלה של e אחרי e' הוא מצב חד ערכי עם 0, כלומר: *איור*
נסמן $e = (p, m)\ ,\ e' = (p', m')$. אם $p \neq p'$ נקבל סתירה ללמת המעויין. נרדים את p, ונביט בסדרה אחרת היוצאת מ־C_1 - המצב בו מתפצלים ע"י הפעלה של e או e' למצב חד ערכי עם ערכים שונים, τ - היא כאמור איננה כוללת את p, ומתישהו מגיעים למצב חד

ערכי. אם זהו מצב חד ערכי עם ערך החלטה 0, סוגרים את למת המעויין עם e ומגיעים שוב לסתירה. אם מחליטים 1, סוגרים את למת המעויין עם $e'e$, ושוב מגיעים לסתירה.

נותר לטפל במקרה בו בריצה ההתחלתית e כן משתתף. במקרה זה, אם אחרי הפעלת e המצב דו ערכי, סיימנו, אחרת המצב הוא חד ערכי עם ערך 0. במקרה זה נעלה למעלה בעץ, וננסה להפעיל את e צעד לפני במעלה העץ, כמו מקודם, ונחפש שוב מצב בו עוברים לערך החלטה 1, ונפעל באותו אופן כמו במקרה הקודם. ■

הבעיה הזו נוסחא ע"י נאס"א בסוף שנות השבעים, והמשפט הוכח באמצע שנות השמונים, אולם רק בשנות התשעים הובנה המשמעות העמוקה של ההוכחה - העובדה שהיא תופסת למודלים אחרים רבים.

כדי לעקוף את FLP עוד לפני שהוכח, אנשים פיתחו אלגוריתמים רנדומיים (חלקם אפילו ישראלים). לא נתמקד מהם, אלא נדון בכמה אספקטים נוספים:

$Two - Face\ Commit$ 2.5.2

מהו פרוטוקול זה?

ב-DB לכל טרנזקציה יש $coopdinator$ המפיץ לגורמים השונים את הטרנזקאציה, מקבל ok או $abort$, מחליט $commit/abort$ ושולח זאת לכולם.

נשים לב כי הוא עומד בתנאי FLP - ההסכמה בסוף היא על $commit$ או $abort$. כשעותקים מקבלים את הטרנזאקציה הם נכנסים למצב נעול, ונתקעים עד לקבלת תגובה מה-$coordinator$. אם הוא נופל, אחרי זמן ארוך כולם מניחים $abort$. יש פיתוחים של הפרוטוקול הזה על מנת למנוע כמה שיותר

2.5.3 בעיית $Paxos$ - החלשת ההסכמה

הרעיון הכללי: מנסים להגיע למצב בו ניתן להגיע לבצע שינויים, כל זמן שכל אחד שיגיע מאוחר יותר יוכל ללמוד על השינוי הזה. זה לא יהיה הסכמה במובן המלא של המילה, אבל ניצור היסטוריה של הסכמות לגביהן נוכל לומר שאם המערכת מתייצבת כך שיהיה אפשר לדבר עם רב המעבדים, אז בשלב הזה "משהו טוב יקרה".

הבעיה הספציפית שאנחנו נציג היא הבעיה הבאה:

ישנו תא בזיכרון שנרצה לשייך אליו ערך (בהמשך נכליל לסדרת תאים שכזו), כך שאם נשייך לו ערך סופית, כל מי שבעתיד ינסה לשייך לו ערך, ילמד על הערך שכבר שוייך, ולכן לא יכתוב שוב (כלומר לא "ידרוך" עליו).

התהליך הזה הוא ליבה של הרבה פרוטוקולים אחרים, וזוקק ככזה ע"י לסלי למפרט: ישנו כפר דייגים - פקסוס. כשרב הדייגים נמצאים בכפר, צריך שיגיעו להחלטה. הוא הניח שברגע יש פרוטוקול שיכול לבחור מנהיג מידי פעם (זוהי חולשה של הפרוטוקול שלו). אזי, אפשר להגיע לרצף החלטות, כך שאם מנהיג אחר ינסה בהמשך להגיע להחלטה באותו הנושא, הוא יגלה שכבר הגיעו להחלטה בנושא.

מה הבעיה?

קיימת סדרת מעבדים, כל אחד יכול לדעת מה ההצעה האחרונה לערך, ואם נגיע לכך שמירב המעבדים ידעו על ההצעה האחרונה, זו תתקבל. כלומר, כל מעבד שומר לעצמו כמה ערכים שנקבע בהמשך, ביניהם v הערך המוצע. כל מנהיג חייב לשאול את כולם האם כבר הוצעה כבר איזושהיא הצעה. אזי אם יש לו גישה לרובם, יתכן וילמד כי כבר יש הצעה - הוא יכול לדעת שמשהו קרה או לא קרה.

הקושי: אם קיימים שני מנהיגים ה"מדברים" בו זמנית, אין אף ערך שניצח. צריך מנגנון כך שאם יש בלאגן המנהיג ידע על כך, ויוכל להציע ערך אחר. כל מנהיג לא יודע אם

הוא מנהיג יחיד או לא, לכן צריך לקבל פידבק על הערכים במערכת כרגע. צריך דינמיקה אפשרית שתדאג להפטר מהצרות.

מה צריך עוד לשמור כדי לבצע זאת? למשל, כל ערך שנשמר צריך משהו שיציג את שם המנהיג שהציע את הערך, או $counter$ שיוצמד לו, ועל פיו נוכל לדעת סדר זמנים בין הערכים: (v, c).

המטרה: להגיע למצב יציב, דהיינו מצב בו אם יש מנהיג שחושב שהצליח וקיבל ok מרב המעבדים לגבי ערך מסויים, ערך זה לא ישתנה בעתיד.

איך עושים את זה?

פרוטוקול ראשוני אם אני מנהיג:

- אסוף מכולם (v_i, c_i).
  - אם יש רוב ל־v', הפץ (v', c') כאשר $c' > \{c_i\}$.
  - אחרת, הצע (v'', c') עבור v'' כלשהוא ו־$c' > \{c_i\}$.

למשל, בפעם הראשונה שהמנהיג ינסה, הוא לא יקבל אף ערכים, ולכן יבחר ערך כלשהוא.

נניח שיכולים להיות קאונטרים אינסופיים. למשל אפשר לבחור v'' ערך התחלתי של המנהיג, $c' = max\{c_i\} + 1$.

חולשות של הפרוטוקול:

- מחכים לתשובות מכולם - נחליף אם כך ברוב.
- אם לא מחכים לכולם, יתכן ואני שמעתי מחלק, ולאלו שלא שמעתי מהם יהיה קאונטר גבוהה מזה שלי.
- מה קורה אם יש יותר ממנהיג אחד? אם יש למשל שניים בהתחלה, אז הם שניהם בו"ז מציעים שני ערכים שונים, ואז לכולם יהיו ערכים שונים עם אותו הקאונטר.

על כן צריך להוסיף לפרוטוקול אפשרות לחזור בנו\משהו בסגנון זה, כלומר המנהיג צריך לברר אם הצליח או לא, כדי שיוכל להתחיל מחדש.

נסתכל על שלוש קבוצות של מעבדים, יתכן ויש חפיפה ביניהן:

- מנהיגים
- שומרי חותם
- לומדים

באיזה שלב נעשה את זה?

c' הוא איזשהוא קאונטר שנרצה שיהיה גדול יותר מהקאונטרים האחרים במערכת. אפשר מראש למשל לנחש מהו c', ואז כשמבקש מכולם את הנתונים, יכול לבקש את התשובות עם הקאונטר המבוקש, ואם מישהו יודע על קאונטר גבוה יותר, הוא ישר יכול לשלוח למנהיג $abort$ (יכול להיות והודעה זו עצמה תגיע מאוחר). יש כאן כמה פאזות בהן אפשר לקדם - מברירם אם הקאונטר טוב, במקביל "זוכים בממלכתיות" - אם יש שני מנהיגים, אחד מהמעבדים ישמע מאחד לפני השני, ואז ישלח $abort$ למנהיג עם הקאונטר הכי נמוך. יתכן וגם שני המנהיגים ישמעו $abort$, אבל זה לא מפריע לנו - הם יתחילו שוב את התהליך. אין בעיה בהצגת ריצה אינסופית, כי אנו יודעים שבכל מקרה יש כזו. נסתמך על כך, כל עוד נוכל להגיד שכאשר תנאים מסויימים יקרו, ערך יקובע.

פרוטוקול שני שומרי החותם שומרים את הערכים הבאים:

- $max\{c\}$.
- (v,c) ערך מוצע.

אם אני מנהיג:

- בחר c'.
- שלח לכולם הודעת בירור עם c'.
- קבל מרוב (c_i,v_i). יהי $\overline{c}$ המקסימלי ביניהם, ו־$\overline{v}$ הערך הצמוד לו. אם לא קיבלת $abort$, שלח $(c',\overline{v})$ לכולם. אם $\overline{v}=0$, בחר את הערך שלך והצע אותו. אם קיבלת $abort$, חזור להתחלה. אחרת ok וסיים.

אם אני שומר החותם:

- בקבלך הודעת בירור עם c':
  - אם $c'\leq max\{c\}$, שלח $abort$.
  - אחרת $max\{c\}=c'$, ושלח (v,c).
- (תוספת מאוחרת יותר) בקבלך ערך $(\overline{v},\overline{c})$:
  - אם $\overline{c}<max\{c\}$, שלח $abort$.
  - אחרת $(v,c)=(\overline{v},\overline{c})$ ושלח ok.

המנהיג עדיין לא יודע אם הפרוטוקול שלו הסתיים ־ כעת כשהוא מציע את הערך $\overline{v}$, יכול והיה מנהיג אחר שהציע קאונטר יותר גדול. נרצה שבמקרה זה המנהיג המקורי יקבל $abort$. לכן נוסיף את השלב השני לפרוטוקול של שומר החותם. 24/12/2012

היופי בפרוטוקול הזה ־ דרישות הסינכרון שלו הן מינימליות ־ אם יש מנהיג זמני מספיק זמן, ניתן להגיע להחלטה.

נשים לב ־ מנהיג מקומי מנסה בעצם "לנצח" כדי להיות המנהיג הנבחר. הוא בוחר לעצמו מונה כלשהוא c, ומנסה לברר מהו מצב הרשת. במידה והוא היה ביזנטי, הוא יכול תמיד היה לבחור באינסוף, ובכך "לדפוק" את המערכת ולגרום לנו להגיע לאי הסכמה. נניח שלא כך המצב.

כאשר מנהיג מקומי מקבל מרב מסוים $\perp$, הוא יבחר ערך כלשהוא שהאפליקציה תרצה לקדם. בעצם המנגנון שהוצע בשיעור הקודם מבטיח שאם ישנו ערך שהוצע למספיק אנשים, הוא יהיה זה שיקודם.

בעיה: שמירת ערך יחיד לכל מונה שומרי החותם משמרים את מצב המערכת כפי שהוא ידוע להם. מהו מצב זה? ערך ההתחלה הוא $\perp$ ומונה כלשהוא בממלכה של c.

אם התהליך הסתיים בהצלחה, יתקיים $max\{c\}=\overline{c}$. נרצה שהערך שיימצא אצל שומרי החותם יהיה ערך יחיד, בהנתן העובדה שערך מסויים מתקבע.

התהליך שאנו עושים הוא איסוף של רב הערכים (המערכת כאמור היא אסינכרונית ולכן לא בהכרח נדגום את כל הערכים האפשריים), ומתוכם בוחרים ערך מסויים. ננסה למנוע מצב בו שני מנהיגים יציעו את אותו מונה, כלומר על אותו מונה c לא יהיו במערכת שני

ערכים שונים. דוגמא אפשרית להיווצרות מצב שכזה הוא מצב בו אחד המנהיגים התחיל לשלוח את הערך שלו לחלק משומרי החותם ונרדם לפני שהספיק לשלוח לכולם. לאחר שהראשון נרדם, מנהיג שני עם ערך אחר מתעורר, וכך עם אותו מונה שולח ערך אחר שמגיע לאותם שומרי חותם שהראשון שלח אליהם, ואולי גם לאחרים. מצב זה בעייתי, שכן יווצרו התנגשויות - לא נדע איזה מהערכים לבחור במקרה והמונה שנשלח הוא מקסימלי (כמו כן כנראה שמצב זה יצור בעיות נוספות בהמשך).

איך נמנע מצב זה? בעזרת ה-id של המנהיג! מספר זה הוא כאמור מספר שמובטח לנו כי הוא ייחודי לכל מחשב.

נגדיר פונקציה חח"ע $g : ID \to \mathbb{N}$, ונגדיר את המונה הבא של המנהיג, c, להיות:

$$f(c) = c \cdot g(id)$$

דרך אפשרית אחרת הינה לבחור את המונה בתור הזוג הסדור (c, id), ואכן צימוד שכזה יהיה יחודי, אפילו אם המונה המקורי c אינו.

בעזרת שיטות אלה נקבל את הרצוי - אצל כל שומרי החותם עם אותו המונה, נשמר אותו הערך.

מדוע בסוף התהליך ישנה הסכמה נרצה להוכיח כי כאשר מנהיג החזיר ok, אזי אצל כל שומרי החותם קיים הערך שלו, כלומר ישנו ערך קבוע במערכת.

נניח כי מנהיג $\overline{p}$ קיבל מרוב שומרי החותם ok (כלומר יש לפחות $\left\lfloor \frac{n}{2} \right\rfloor + 1$ כאלו שאישרו). נביט במנהיג p' שהתחיל אחרי השלב בו $\overline{p}$ סיים ויעבור את השלב הראשון בפרוטוקול. אזי המונה c' שהוא צריך להציע הוא לפחות $\overline{c}$ - המונה של $\overline{p}$ (שכן רב שומרי החותם כרגע מקיימים $max\{c\} = \overline{c}$). ברגע שהוא יסיים את השלב הזה, הוא יהיה חייב לראות את הערך $\overline{v}$ (הערך של $\overline{p}$), ולאף אחד אחר אין $v' \neq \overline{v}$.

נניח כי יש שומר חותם המסיים עם (v', c') כאשר $c' > \overline{c}$, ונראה כי במקרה זה בהכרח $v' = \overline{v}$:

תחילה, נציין כי כל שומרי החותם ששלחו ל-$\overline{p}$ הודעת ok לא שמעו עדיין מ-p', שכן במקרה זה היו שולחים ל-$\overline{p}$ הודעת $abort$ בזמן בקיבוע (שכן $c' > \overline{c}$). אזי, איך ייתכן שמישהו מחזיק בזוג (v', c')? במקרה זה, היה מנהיג שהצליח לקדם את v' כל הדרך, כלומר הוא קיבל ok בשלב הראשון מרוב שומרי החותם, אשר ביניהם קיים לפחות אחד מאלו ששלחו ל-$\overline{p}$ הודעת ok, לכן הוא יבחר את הערך המקסימלי שהוא קיבל - כלומר את $\overline{v}$, ולכן בשלב הקבוע הוא ישלח לכולם $(\overline{v}, c')$.

כאן אנו מסיימים עם פרוטוקול זה - השתמשנו בו לקביעת ערך אחד, אולם ניתן לקבוע סדרה של ערכים ולבצע דברים נוספים בעזרת פרוטוקול זה.

2.6 *Approximate Agreement*

מוטיבציה

בכל מה שהוצג לנו עד כה תמיד נדרשה הסכמה, כלומר נדרשנו להחליט על ערך כלשהוא. עם זאת, לעיתים קרובות כל מה שנדרש הוא לא הסכמה מדויקת (כי לרב במצבים אמיתיים לא ניתן לעשות זאת), אלא דבר אחר:

במידה והמעבדים מחזיקים ערכים שונים, נרצה שהערך הסופי של כל הטובים יהיה איזושהיא פונקציה של הערכים שלהם, כמו ממוצע משוקלל או פונקציה אחרת. במקרה שכזה לא נוכל להחליט על ערך $\perp$.

למשל, דגימת טמפרטורה ע"י חיישנים:

כאשר דוגמים חיישנים שונים, הטמפורטורות שידגמו יהיו שונות זו מזו (מכל מיני סיבות), נגיד בחצי מעלה. כמובן שאין זה אומר שאין מדידה נכונה (המדידה היא מדוייקת). מכיוון וכל החיישנים מדדו טמפרטורה באותו הטחח, נעדיף במקרה זה לבחור את ממוצע הטמפרטורות שנמדדו.

דוגמא נוספת יכולה להיות סנכרון שעונים - נרצה שכל השעוים יהיו מספיק קרובים זה לזה.

נציג את הבעיה באופן פורמלי, ופרוטוקול שיספק אותו:

הגדרת הבעיה

בסוף הפרוטוקול יתקיים:

- כל שני טובים מחזיקים ערכים עד כדי ε זה מזה בסוף הפרוטוקול.
- הערך של כל טוב הינו בטווח הערכים ההתחלתיים של כל הטובים.

אם נענה על התנאים הללו נחליש את ההסכמה, אך עדיין כפי שציינו יש לכך שימושים.
ההנחות שלנו:

- נניח בהתחלה סינכרוניות עם ביזנטיות.
- נניח $t < \frac{n}{5}$.

פיתוח פרוטוקול

רעיונות:

- כל אחד ברגע שהוא מקבל ערכים מכולם יבחר איזשהוא טווח ביניים, עליו נפעיל את הפונקציה של ממוצע משוקלל (זאת בכדי להתעלם מערכים רעים).
- מכל ההודעות שנקבל, נזרוק t הודעות מההתחלה ומהסוף, כך שנשאר עם $\#messages > \frac{3n}{5}$.

רעיון זריקת הערכים עוזר בצמצום טווח הערכים האפשריים לנו - בפרוטוקול שנציע, בכל פעם נצמצם את טווח הערכים האפשריים בחצי, וכך נוכל להמשיך עד לקבל ה-ε הרצוי.

ניתן להשתמש ברעיון זה לסנכרון שעונים, אך למטרה זו נבחר רעיון אחר המתכנס מהר יותר?

סנכרון שעון

מהו סנכרון שעון?

נניח שישנם שני מעבדים - M, M'. שעון הוא בעצם אלמנט חומרה אותו אנו דוגמים במעבדים. שעונים במעבדים שונים מיוצרים בטכניקות שובות, ובעלי דיוק מסויים + זליגה (אלא אם כן השעון הוא אטומי). מידת הזליגה מושפעת מגורמים שונים, ביניהם למשל חום.
השעון למעשה מורכב משני מרכיבים:

$$C = C_p + C_l$$

כאשר C_p הוא שעון החומרה הפיזי, ו-C_l הוא תיקון לוגי.

לרב נניח כי קצב הזליגה של השעון הפיזי הוא $10^{-8}/10^{-5}$. עם זאת, אותנו מעניין התיקון הלוגי - נרצה למצוא אלגוריתם שימצא את התיקון שיש להוסיף על מנת שלכל הטובים יהיה ערך קרוב כמה שאפשר אחד לשני.

נניח שסינכרנו את כל הטובים שברשותנו. עדיין יתכן מצב בו שעון אחד מתקדם יותר מהר מהשני, ולאט לאט הפער ביניהם יגדל. במצב זה נריץ שוב את אלגוריתם הסינכרון.

זהו בעצם אלגוריתם איטרטיבי, שנאלץ להריץ כל תקופת זמן. אם נצטרך להוריד מקצב השעון הפיזי (כלומר להוסיף תיקון לוגי שלילי), נוריד את קצב התקדמות השעון הפיזי, ובכך בעצם "נאיט" את הזמן שנראה מהשעון הפיזי (למשל ע"י התעלמות ממספר סיבובי שעון).

נסמן ב-$\rho \simeq 10^{-4}$ את קצב זליגת השעונים - אם הזמן האמיתי הוא t, אזי הערך שהמעבד יכול לראות הוא $\pm\rho \cdot t$. נשים לב שאם נסמן זמן שנקבל מאחד השעונים האחרים שנדגום ב-t', ונסמן ב-d את אי הוודאות לזמן שבו נקבל את ההודעה, אזי הזמן t' הוא בטווח $t' \pm d$.

הערה 2.15 כשנרצה לסנכרן שעונים נניח שישנו מאסטר ועבד (Master and Slave), כאשר המאסטר מחובר לשעון חיצוני והוא מסונכרן תמיד, ואת העבד נרצה לסנכרן.

נניח שהמאסטר שולח לי הודעה שהזמן אצלו הוא T_1, וברגע זה אצלנו הזמן הוא T. אם לאחר מכן הוא ישלח לי זמן T_2, והזמן אצלנו הוא T', וכן הלאה, לא נוכל לתקן את הזמן, כי אנחנו לא יודעים מהו ההפרש. אם נוריד כל הזמן d (טווח אי הוודאות), אז אנו יכולים גם לחרוג מהזמן האמיתי (שכן לא בהכרח לוקח להודעה להגיע d בדיוק, זהו רק חסם עליון). ישנן שלושה גורמים שיכולים להשפיע:

1. המידע המחכה לשליחה אצל השולח.
2. המידע על הקו (או הפרעות).
3. כמה מידע מחכה להכנס אצל המקבל.

אם ניצור מצב של פינג-פונג, כלומר אם נשלח הודעה T' למאסטר ונקבל ממנו חזרה את ההודעה T_1 בזמן T'', נוכל לדעת, אם נניח למשל שההודעה היתה צריכה להגיע למאסטר במחצית הזמן שעבר, מה השגיאה שלנו ביחס לזמן שקיבלנו. כלומר, נוכל למצוא את אלמנט התיקון כך:

$$C_t = T_1 + \frac{T'' - T'}{2}$$

אסטרטגיה נוספת היא שנוכל לשלוח $burst$ של הודעות, ולבחור את אלמנט התיקון המינימלי מבין כל האלמנטים שיחושבו, ובכך לשפר את האלמנט הלוגי.

הנחות

- נניח שלכל המעבדים יש שעון, ולא בטוחים באמיתותו.
- נניח כי חלק מהמעבדים הינם רעים (יכולים להריץ את הטובים מהר יותר, או לעכב אותם ע"י שליחת זמנים לא נכונים).
- קיים חסם d המכיל את כל האי וודאויות שיש לנו בשליחה בין הטובים - המערכת הינה סמי-סינכרונית.
- ישנן חתימות דיגיטליות.

רעיונות

- שימוש ב־*Approximate Agreement*.
- נחפש עוגן, לדוגמא: אם אנו יודעים שאמור להגיע שעה אחרי או לפני הזמן שאנחנו מאמינים בו, נקבל הודעה T אמ"מ ההודעה מקיימת $T = T_{mine} \pm d$.
- נניח קיום d_{max} המקסימום לפני הסנכרון של המרחק בין השעונים הטובים, אז כל מעבד ששולח לי הודעה עם זמן T המקיימת $T < T_{mine} \pm d_{max}$ (או גדול וכו') נדע שהוא רע.
- אם נקבל הודעה ממעבד החתומה ע"י שני מעבדים, והזמן המקורי קטן משלנו, נוכל להקפיץ את הזמן של המעבדים בשרשרת.
- אם נקבל הודעה עם $t + 1$ חתימות, לא נחתום, שכן קיים איזשהוא טוב שראה את אחת ההודעות.
- נגדיר δ בתור המקסימום בין קריאות השעון של הטובים במהלך הפרוטוקול.

ישנו עוד תחום (בו לא נעסוק במסגרת הקורס) הנקרא gradient clock syncronization ־ הטובים מחוברים אחד לשני (הרשת לאו דוקא מלאה), והמטרה היא לסנכרן בין כולם. נרצה לסנכרן כל מעבד לסביבה קרובה שלו, כלומר סינכרון שלי יהיה ברמת דיוק הכי גבוהה לשכנים הכי קרובים, וברמת דיוק נמוכה לשכנים הרחוקים, דהיינו מידת הסנכרון שלי ממעבד תלויה במרחקו ממני. מסתבר שבמקרה שכזה לא תלויים רק ב־d, אלא גם בקוטר הגרף של הרשת.

3 אבסטרקציה של בעיות כלליות

31/12/2012 (חומר מהספר של חגית עטיה)

היום נעזוב בצד את הפרוטוקולים של בעיות פרטיות, וננסה לעשות אבסטרקציה של פתרון של בעיה אמיתית. כאן נדון בעיקר בעולם אסינכרוני (אולי נחזור לסינכרוניות מלאה או חלקית בהמשך.

פתרון בעיה: הגדרה של סדרות קלט וסדרות פלט, והיחסים ביניהם.

למשל בעית ההסכמה - אם מסתכלים על קלט מסויים למנהיג, הוא נותן תנאי לפלטים שצריך לקבל אצל כל המעבדים האחרים.

הפרוטוקול שלנו הוא למעשה יישום של "קופסא שחורה", שמתחתיה נמצאת הרשת בהתאם למודל שלה.

למה נרצה להגידר מהו פתרון בעיה? כדי שנוכל להחליף את הקופסא השחורה הזו בקופסאות אחרות, ששומרות על הגדרת הבעיה.

נרצה להסתכל על מצב בו יש מערכת, יש לה קלט ופלט, שולחים הודעה לרשת התקשורת ומקבלים הודעות חזרה. ההודעות שנקבל חזרה "יושבות" בקופסא עד שמחליטים מהו הפלט. נוכל לקחת את הקופסא הזו ולחלק אותה לשכבות (לא קל עבור כל "קופסא שחורה" שכזו), כאשר כל שכבה בעצם מקבלת מהשכבה שמעליה איזשהוא קלט, מדברת עם קלטים ולפטים עם השכבה שמתחתיה, ומוציאה פלט לשכבה שמעליה.

כך למעשה אפשר להסתכל על שכבות שונות בקופסא, כאשר כל שכבה שכזו מוגדרת על ידי אוסף ההודעות והיחסים שביניהם. כך אפשר להחליף בין השכבות, כאשר הקופסא השחורה הכוללת עדיין תפתור את הבעיה.

היום נראה כיצד לחלק לשכבות ולהחליף ביניהן, כך שעדיין יתקבל פתרון לבעיה הכללית.

כך נראה למשל איך פרוטוקול אחד מסמלץ פרוטוקול אחר: נניח כי נתון פרוטוקול C וההפרוטוקול C' רוצה לסמלץ אותו. מה זה אומר? בהנתן אותה סדרת קלטים, מתקבלת אותה סדרת פלטים, כאשר ההנחה היא שמי שמסתכל רואה רק את סדרות הקלט והפלט. נראה דוגמאות שכאלה בהמשך (כנראה בשבוע הבא).

3.1 אבסטרקציה ראשונה - $broadcast$

במערכת א־סינכרונית אפשר להגדיר מה זה ברודאקסט - יחולק לשני חלקים:

- $bc - send_i(m)$: i קיבל קלט m ושולח אותו בברודקאסט לכולם.
- $bc - rec_i(m_j)$ משלימו בפלט - ההודעה m נשלחה ע"י j והתקבלה ע"י i. נשים כרגע שגיאות ביזנטיות בצד, ונראה כיצד אפשר לפתח את הדוגמאות שנדון בהן.

את האבסטרקציה של ברודאקסט כנ"ל ניתן לממש בדרכים שונות, אולי עם תכונות נוספות שנרצה להוסיף בהמשך.

השאלה כעת היא מהי היחסים בין סדרות הקלט והפלט במכונות השונות? למשל, אפשר לדרוש שהודעה תתקבל רק פעם אחת - כלומר אפשר שיתקבלו כמה עותקים, אך רק עותק אחד יעבור הלאה.

אילו תכונות לדרוש ממערכת שכזו?

- $integrity$ - כל עוד הודעה שהתקבלה אכן נשלחה.
- $no\ duplication$ - כאשר מקבלים הודעה, היא תתקבל רק פעם אחת.

- $liveness$ (כאשר אין שגיאות) - יש מיפוי בין הקלטים לפלטים, כלומר כל מה שנשלח, מתישהוא יגיע (לכולם במקרה שלנו של ברודקאסט).

מימוש אפשרי שליחה $point-to-point$: משלוח של מעבד אחד למעבדים מסויימים שהוא מגדיר.
נבצע את השליחה בצורה זו - ניקח את ההודעה ונשלח ב־$point-to-point$ לכולם.
בעת קבלת הודעת $point-to-point$ מציפים אותה לכולם, בתוספת של מי שקיבלנו ממנו את ההודעה.
כאן אנו מניחים שאנו יודעים מי נמצא ברשת. יש מערך שלם של מחקרים בנושא של תקשורת קבוצתית. זה קשור למה שנדבר היום, רק מוסיף לזה עוד נדבך בו עושים כל הזמן $maintanence$ של "מיהם חברי הקבוצה". איך יודעים מי נמצא ברשת? יתכן וזה נעשה ע"י שכבה נוספת, אצלה כולם נרשמים ומבצעת את התחזוקה של הרשימה.
מימוש זה מקיים את הדרישות:

- $integrity$ - כל הודעה שהתקבלה באמת נשלחה.
- $no\ duplication$ - כי שולחים רק פעם אחת.
- $liveness$ - בהנחה שהרשת אכן אמינה, כל הודעה בסופו של דבר תגיע ליעדה ולכולם.

אפשר עכשיו למשל להרחיב את המושג הזה - הגדרנו עבור הודעה בודדות, אפשר לדרוש קופסאות עם תכונות יותר חזקות. תכונות שתעניינה אותנו ידברו על היחסים בין הקלטים והפלטים בין המכונות השונות, או היחסים בתוך מכונה מסויימת.
אפשר למשל להסתכל על קופסא היושבת מעל הקופסא הזו, עם $bc-send_i\left(m,qos\right)$, $bc-rec_i\left(m,qos\right)$, כאשר qos יכול לקיים אחת מהתכונות הבאות:

- $basic$ - ההגדרה הראשונה שלנו.
- $fifo$ של שולח - סדרת הפלטים שמשוייכים להודעות שיוצאות מאותו המעבד, יהיו באותו הסדר אצל כל המקבלים. כאשר מממשים מעל רשת א־סינכרונית, זה דורש שמירה של ההודעות עד שמגיע הזמן "להעלות אותן למעלה".
- סידור קוזאלי - יותר חזק מהקודם, יותר חלש מסידור מלא, אך מסובך יותר להסביר. אינטואיטיבי רוצה לומר: מגיע בסדר השומר על תלות אפשרית בין ההודעות. באופן פורמלי נגיד ש־m_1 קודם קוזאלית ל־m_2, ונסמן $m_1 \alpha m_2$, אם מתרחשים אחד התנאים הבאים:
  - m_1 נשלחה לפני m_2 מאותו מעבד.
  - m_1 התקבלה במעבד לפני ש־m_2 נשלחה ממעבד זה.
  - יש m_3 כך ש־$m_1 \alpha m_3 \alpha m_2$.
- סידור מלא - ההודעות מתקבלות באותו הסדר אצל כל מעבד, כלומר כל סדרה שמתקבלת היא תת סדרה של מעבד אחד (בגלל הא־סינכרוניות לא ידוע מתי הכל יתקבל).

רק מלא לא בהכרח $fifo$, אבל מלא ו־$fifo$ הוא קוזאלי. יש סדרים קוזאליים שאינם בהכרח מלאים.
נראה שאפשר לממש כל אחת מהשכבות הללו מעל qoc_{basic}:

- $fifo$ אצל השולח - השכבה תצרף $counter$ לכל שולח, ולכל אחד מעביר רק לפי ה-$counter$ המתאים לו. אז אין בעיה למימוש, כי שלושת התכונות שרצינו נשמרות, ואם אין שגיאות וכן אם כולם מתחילים יחדיו באמת מתקבל $fifo$.
- סידור מלא - דורש כתיבת פרוטוקול וכבר אינו כל כך מיידי...

3.1.1 פרוטוקולים אפשריים למימוש סידור מלא מעל q_{basic}

כבר יש לנו את קופסאת ה-$fifo$ מעל ה-$basic$. איך נממש מעליה סידור מלא? צריך להשתמש במשהו שמקביל להסכמה, או מספר דרכים קלות יותר - הבעיה היא שאי אפשר לסמוך מראש על הכתבת הסדר (כי אם מישהו לא מקבל הודעה בטווח זמן מסויים הוא יתקע לנו את כל המערכת).

הפתרון הכי טריוויאלי - יש רק אחד שקובע את הסדר. איך ממשים? כל מעבד שולח ב-$point-to-point$ את כל ההודעות למעבד מוסכם (אם יש מנהיג זה קל), והוא שולח ב-$fifo$ שלו הודעה לכולם.

דרך נוספת - כל אחד שולח לכולם ב-$braodcast-basic$, ומחכים לקבל מהמנהיג את הסדר - $counter$ להעברה למעלה. כך המנהיג לא צריך לשלוח ב-$broadcast$ נוסף את כל ההודעות, ואז ניתן לחסוך בתקשורת (יש סביבות מסויימות בהן דרך זו עדיפה).

פרוטוקול סימטרי נרצה להציג גם מימוש סימטרי - כל המעבדים יריצו את אותו הפרוטוקול. נסמן את $tbc-send$ להיות פעולת השליחה בפרוטוקול זה, ו-$tbs-rec$ להיות פעולת הקבלה.

מהו הקושי בסדר מלא? אנו יודעים ממי הגיעה כל אחת מההודעות. יתכן ויהיה מידע נוסף להודעה עליו נחליט בהמשך. מה צריך לדעת על מנת שיהיה אפשרי לשלוח הודעה? תחילה, אנו צריכים לדעת את המצב אצל המעבדים האחרים: למשל אם הגיע ה"תור" של מעבד, אולם הוא לא שלח הודעה, נוכל להתקע לנצח, כלומר נצטרך לדעת שלא שלח.

כיוון אחד - הסכמה על כל הודעה. אבל זה עם $overhead$ מאוד גדול.

כיוון שני - לשלוח הודעה "פיקטיבית" ב"בטן" של ה-$total$ בעזרת $counter$ - נמספר הודעות ונעלה את הקאונטר כל פעם שנשלח, ונשלח הודעה פיקטיבית אם אין הודעה. איך נחליט כיצד להעביר הודעה?

נסתכל על גל ההודעות הראשון - כולם מתחילים עם קאונטר אפס - נעביר הודעה ונשלח לכולם. איך נדע את מי להעביר הלאה?

אפשר להסתכל על הזוג הסדור $\langle c, id\rangle$. האם אפשר להחליט על הזוג הסדור הקטן ביותר? לא! אולי עדיין לא התקבל אצל מישהו, ואז אסור לשלוח.

אם הועלתה הודעה מה-$fifo$ ל-$total$ - כאשר מקבלים קאונטר פיקטיבי, אנו יודעים שיש לנו כבר את כל ההודעות בעלות קאונטר קטן יותר שנשלחו ע"י המעבד הזה.

מתי נדע שאפשר להעביר את ההודעה $\langle 1, id\rangle$ עם ה-id המינימלי? אם קיבלנו מכולם הודעה עם קאונטר גדול יותר מאחד. אז "יש לנו בשק" את כל ההודעות הקודמות. נשמור וקטור עם הקאונטר המקסימלי שקיבלנו מכל אחד. אזי אפשר לדעת שכל ההיסטוריה עם קאונטר קטן מזה כבר התקבלה, ואז אפשר להעביר את המינימלי.

מתי יודעים להחליט לשלוח קאונטר פיקטיבי? כאשר קיבלנו הודעה ממישהו עם קאונטר גדול יותר ממה שלנו יש כרגע.

ננסה לכתוב את הרעיון באופן פורמלי, ונראה שהוא אכן ממש את כל התכונות שנרצה שיקיים:

$P_i - tbc$ קיים מערך מונה c עם ערך התחלתי $c[i] = 0$.

1. בקבלך הודעה m לשליחה:

$$c[i] := c[i] + 1$$
$$bc - send_i(\langle m, c[i] \rangle, fifo)$$

2. בקבלך הודעת $bc - rec_i(\langle m, v \rangle, j, fifo)$:

   (א) צרף (m, v) ל־M (אוסף ההודעות שמחכות)[4].

   (ב) $c[j] := v$.

   (ג) אם $v > c[i]$ אז $c[i] = v$, ושלח לכולם $b - send_i(\langle c[i] \rangle, fifo)$ (הודעה "פיקטיבית" - ריקה, רק הקאונטר).

3. בקבלך הודעה $bc - rec_i(\langle v \rangle, j, fifo)$:

   (א) $c[j] := v$[5].

4. יהי m כך ש־$\langle m, v \rangle \in M$ (כלומר ההודעה מחכה להעברה), ו־$\langle v, j \rangle$ מינימלי ב־M, ו־$v \leq c[k]$ לכל k, אז: $tbc - rec_i(m, j)$.

יש להוכיח את שלוש התכונות הנדרשות, וכי הסידור המלא נשמר.

טענה 3.1 שלוש התכונות הדרושות נשמרות.

הוכחה: אין לנו בעיה עם שלוש התכונות הנדרשות, שכן בצורה ברורה הפרוטוקול שלנו עושה זאת - הוא פועל מעל $fifo$ שמקיים את התכונות האלה. נותר רק להראות כי אין הודעה ש"נתקעה" ב־M.

נניח בשלילה שקיימת הודעה שנתקעה ב־M, נסמנה ב־$[\langle m, v \rangle, k]$. אם היא "נתקעה", אז תחילה ה־v לא מינימלי. נניח אם כך כי זוהי ההודעה עם ה־v המינימלית בשק. אם לא הצלחנו להעביר אותה, קיים j כך ש־$v > c[j]$. ההודעה הזו $\langle m, v \rangle$ מתישהו תגיע ל־j בגלל ה־$fifo$, ואז או שהוא יצור הודעה פיקטיבית ב־3, או שיעלה את הקאונטר ב־2, על כן ההודעה הזו לא יכולה להתקע לתמיד, ולכן אין $deadlock$. ■

נותר להוכיח סידור מלא - כלומר :

טענה 3.2 אם מעבד i העביר את $\langle m_1, j_1 \rangle$ לפני $\langle m_2, j_2 \rangle$, זה קורה בכל מעבד.

הוכחה: כאשר i העביר את $\langle m_1, j_1 \rangle$, הזוג הזה היה המינימום בשק שלו. אם $\langle m_2, j_2 \rangle$ כבר היה בשק אזי הקאונטר שלו גבוה יותר, ולכן כולם יעשו זאת גם כן. אחרת הוא לא בשק עדיין, ולכן כשיתקבל הקאונטר שלו יהיה גבוה יותר, וזה גם יהיה נכון עבור כל מעבד. ■

[4] נשים לב כי ה־$fifo$ כאן דאג כי כל הודעה תתקבל רק פעם אחת

[5] לא חייב - ההודעה הזו נשלחה כתוצאה מהודעה שהגיע למעבד אחר - כלומר היא תגיע אלינו בשלב מסויים, ולכן ממילא מתישהו נעדכן

3.1.2 פרוטוקולים אפשריים למימוש סידור קוזאלי - cbc מעל q_{basic}

אם נבנה מעל $total$ שמעל $fifo$, נקבל קוזאלי באופן טריוויאלי (כמו שעשינו קודם), אבל אז היינו חייבים להעביר את כל ההודעות דרך כל השכבות הללו. ב־$total$ היינו חייבים, אבל כאן לא, בשל ההגדרה של הקוזאלי (אחרי קבלת מקומית, אפשר מיד לשלוח אותה - נראה כנראה בשבוע הבא).

אם כך אנו רוצים למצוא מימוש cbc מעל אילושהן שכבות שנרצה. איננו מחפשים את דרגת החופש המקסימלית, אלא משהו שהיה קונסיסטנטי עם קוזאליות, אך לא מחמיר כמו טוטליות (סידור שלם).

באופן אינטואיטיבי ברור כי התכונה הבעייתית היא התכונה השלישית - הטרנזיטיביות. נביט בפרוטוקול עבור $total$ - האם יש משהו שנוכל לעשות בשלב הראשון, שיעזור לנו בשלב האחרון?

רעיון שעלה: לכל הודעה, בנוסף לקאונטר שלי, נצרף להודעה קאונטר קוזאליות - $\langle v,j\rangle$: הקאונטר שיסמן את המקום בו ניתן להעביר את ההודעה (בין $\langle m,c\left[i\right]\rangle$ לבין $\langle v,j\rangle$) - ההודעה האחרונה שהעברנו כלפי מעלה.
מתי נעביר את ההודעה $[\langle m,v,[v_k,k]\,,j,-\rangle]$?

- צריך לוודא ש־$[v_k,k]$ נשלחה קודם לכן. זה מתקבל ישירות אם נשתמש ב־$fifo$.
- בדיקות על v,j כבר יש מ־$fifo$ (לצורך פישוט נדבר על מימוש כזה, אולי נרחיב בהמשך).
- כשמסתכלים על ההודעה הזו, אין בעיה להחליט את מי להעביר. הבעיה היא שזה לא מבטיח את הסגור הטרנזיטיבי של הקוזאליות - מכיוון ויש דרגות חופש בעת העברת הודעות למעלה, הזוג הסגור אצלי אומר מה הדבר האחרון שהעברתי - $[v,j]$: זה הזוג הסגור האחרון שהעברתי, אבל יכול להיות ובחרנו כמה להעביר, וזו היתה האחרונה - ביניהן אין קדימות, שכן הן נוצרו ביחד. כלומר, עלינו למצוא דרך לבטא את שאר ההודעות. על כן בשלב הראשון עלינו לשלוח את כל c, ולא רק את זוג סדור יחיד - זה לא יבטא את כל חזית הקבלה, אלא גם אחרות ששמעתי וכבר העברתי. עם זאת בקוזאליות יש דרגת חופש, ולכן זה יספק אותנו.

לאחר השינוי, יש לנו הודעות בשק מהצורה $[\langle m,c\rangle\,,j,-]$, כאשר c הוא וקטור שלם. איך נחליט את מי להעביר? הבעיה: יש כמה הודעות עבורן אין סדר מלא בין הוקטורים. אזי, בהנתן שתי הודעות $[\langle m_1,c_1\rangle\,,j_1,-]\,,\ [\langle m_2,c_2\rangle\,,j_2,-]$, כאשר $c_1\leq c_2,\ c_2\leq c_1$ (אם $c_1=c_j$, יש ביניהם סדר הכלה מלא), האם יש כאן תלות קוזאלית (ואז לא נוכל "להחליט נכון" באופן שישמר את התלות)? נבדוק את שלושת האפשרויות:

- באותו מעבד לא יכול להיות - כל פעם שקיבלנו הודעה הוקטור עודכן, לכן נשמר יחס הכלה.
- אם היה מעבד שראה הודעה אחת לפני שראה את השניה, אז גם היה סדר בין הוקטורים.
- האם יכול להיות שיש מעבד שלישי שיש לו יחס ל־j_1 ול־j_2, ואז תהיה טרנזיטיביות? אם היה, הקאונטרים היו צריכים לבטא את זה - כלומר וקטור הביניים של m_3 היה נגיד גדול ממש מ־c_1. אם היה m_4 אז הוקטור שלו היה גדול ממש מהוקטור של c_3. כלומר, היה צריך להשמר יחס.

על כן יחס ההכלה מבטא את ההיסטוריה הקוזאלית - דהיינו במצב בו $c_1 \leq c_2 \wedge c_2 \leq c_1$, נוכל לבחור כל הודעה שנרצה ועדיין נשמור על הקוזאליות.

נרצה להחליט אם להשתמש ב־$fifo$ או ב־$basic$ בשליחה. מעל $basic$ יש בעיה. אם היתה הודעה קודמת מעל אותו המעבד, ב־$fifo$ לא היתה לנו בעיה. אם זה קוזאלי מעל מעבד אחר לא מובטח לנו כלום.
נזכור איפה נשארנו - יש כאן מספר פתרונות מעל $basic$ ומעל $fifo$, נחשוב עליהם בבית ונדבר עליהם בשיעור הבא...

7/1/2013

$P_i - cbc$ קיים מערך מונה c עם ערך התחלתי $c[i] = 0$.
פרוטוקול מעודכן:

1. בקבלך הודעה m לשליחה, כלומר $cbc - send_i(m)$:

$$c[i] := c[i] + 1$$
$$bc - send_i(\langle m, c\rangle, basic)$$
$$cbc - rec_i(m, i)$$

2. בקבלך הודעת $bc - rec_i(\langle m, u\rangle, j, -)$, אם $i \neq j$:

   (א) צרף $(\langle m, u\rangle, j)$ ל־M.

3. (3 לא יהיה כאן כמו בפרוטוקול הקודם) מתי מבצעים $cbc - rec_i(m, j)$? כאשר ההודעה היא בראש התור באופן הבא:

   (א) $c[j] = v[j] - 1$.
   (ב) $v(j') \leq c(j')$.
   אז:

$$cbc - rec_i(m, j)$$
$$c[j]++$$

   והוצא את ההודעה מ־M.
אם אין מינימלי יחיד, הוכחנו כי אפשר לקחת כל אחת מהן מבלי לפגוע בקוזאליות.

נראה איך הדברים האלה נשזרים בדברים שנעשה בהמשך: יש פעולות\טרנאזקציות שניתן לבצע מיד, וישנן פעולות שדורשות $blocking$.

איך מוכיחים את נכונות הפרוטוקול? נסתכל על הודעות תלויות קוזאלית, ונראה כי הן הקוזאליות נשמרת:
נניח ובמעבד p_j נעשו הפעולות בסדר הבא:

$$rec(m_1)$$
$$send(m_2)$$

כאשר m_1 נשלח ע"י p_i. כאשר p_j העלה את הקאונטר שלו ל־m_2, הקאונטר ל־m_1 כבר עודכן.
נביט במעבד אחר, ונראה שאינו מעלה את m_2 לפני m_1 - התנאי השלישי בפרוטוקול מבטיח זאת. ה־c המקומי יקיים את התנאי $v(j') < c(j')$ רק אם הקאונטר של m_1 כבר עלה, ולכן m_2 יעלה לפני m_1.

גם כאן מספיק לנו להשתמש ב־$basic$, כי את ה־$fifo$ מקבלים מהקאונטר.

3.1.3 פרוטוקול עם נפילות

עד עכשיו לא הרשנו במימושים שלנו נפילות. הבה נניח שהיתה נפילה. נרצה לממש למשל $bc-send,\ bc-rec$ של $basic-reliable$ (מעל הברודקאסט הרגיל). נרצה להבטיח שאם מישהו קיבל הודעה, כולם יקבלו אותה.
התהליך במעבד p_i:

- בקבלך $bc-send_i(m, rel)$:
  - בצע $bc-send_i(m, basic)$.
  - בצע $bc-rec_i(m, i, rel)$.
- בקבלך $bc-rec_i(m, j, basic)$:
  - אם עדיין לא שלחת את m, בצע $bc-send_i(m, basic)$.
  - בצע $bc-rec_i(m, j, rel)$.

כעת, אם נרצה להרחיב לסדר קוזאלי או מלא פשוט נשתמש במה שבנינו בשבוע שעבר - "נדחף" שכבת $reliable$, וכל יתר ה־$stack$ נשאר אותו הדבר.

3.2 $Shared\ Memory$

3.2.1 הגדרת הבעיה

עד כאן עסקנו רק בהעברת הודעות - ניתן את אותם רעיונות שראינו עד כה לממש גם ב־$shared\ memory$: זוהי סביבה בה מוגדרים אובייקטים (וירטואליים) ופעולות על האובייקט. ההגדרה של הפעולות על האובייקט הן בהנחה שאני בלבד עושה את הפעולה (פעולה סדרתית) - למשל, אם התא הוא תא כתיבה בלבד, הפעולה תתבצע רק ע"י משתמש אחד בלבד עד לסיום הפעולה (סדר הפעולות לא משנה).
ההנחה - לא נעשות פעולות מקומיות בו זמנית.
היינו רוצים לקבל סימולציה של $shared\ memory$ מעל מערכת מבוזרת, בדומה למה שעשינו בשבוע שעבר והיום. כלומר, כל אחד מהמעבדים יכול להחליט איזו פעולה הוא מבצע על איזה אובייקט, וכאשר הפעולה מתבצעת כפי שנגדיר אותה, הסמנטיקה חייבת לקיים מימוש סדרתי.
נסתכל על אובייקטים עם פעולות $read$ ו־$write$ בלבד - זוהי ליבת הבעיה, וניתן להרחיב את המימושים לפעולות נוספות (אך לא נעשה זאת).
כמו כן, נשים לב כי אנחנו צריכים לדעת לוקלית מתי פעולה מסתיימת (כדי לדעת מתי להתחיל את הפעולה הבאה) - כלומר לכל פעולה צריכה להיות תגובה, ולמעשה נקבל מימושית זוגות של פעולות:

- אחרי פעולת $w_p(x)$ מתבצע $ok_p(x,v)$.
- אחרי פעולת $r_p(x)$, מתבצע $ret_p(x,v)$.

נשים לב כי אם שתי פעולות קריאה התרחשו בו זמנית, לא משנה איזו התרחשה קודם, כלומר הסדר שלהן לא משפיע.

נרצה לבטא את תכונות הקונסיסטנטיות הבסיסיות שנרצה לממש בסדרה σ של זוגות של פעולות כאלו (הפעולות המרכיבות זוג כמובן לא חייבות להופיע בסדרה אחד לאחר השני מיידית):

- נכונות - כל מעבד לכשעצמו, הפעולות שהוא רואה לגבי האובייקט ($read, write$) שומרות את הפעולות הסדרתיות על האובייקט. כלומר, מנקודת מבט של מעבד p_i על הסדרה - $\sigma|_i$, היא מכילה זוגות "תקפים" של פעולות על האובייקט (כאילו נעשו באופן סידרתי).
- $liveness$ - ב־σ כל פעולה מסתיימת (כלומר σ באמת מכילה זוגות של כל הפעולות שבוצעו).
- לינאריות - כאן נציין את כל מה שנרצה לקבל מהמימוש של האובייקט:
  - $\sigma|_o$ מבטאת פעולות סדרתיות תקפות על האובייקט.
  - יש פרמוטציה π של σ כך ש:
    * קונסיסטנטיות מקומית - אם o_1 הסתיים ב־p_i (כלומר זוג פעולות בודד המגדיר את הפעולה $read$ או $write$ על מעבד נתון) לפני ש־o_2 התחיל, אזי o_1 לפני o_2 ב־π (כלומר אפשר לסדר את σ בסידור ששומר סידרתיות).

דוגמא: נניח וישנם שני אובייקטים X, Y, שני מעבדים p, q, וסדרת הפעולות:

$$\sigma = w_p(X,1)\ \ w_p(Y,1)\ \ ok_p(X)\ \ ok_q(Y)\ \ r_p(Y)\ \ r_q(X)\ \ ret_p(Y,1)\ \ ret_q(X,1)$$

פרמוטציה π מתאימה יכולה להיות:

$$w_p(X,1)\ \ ok_p(X)\ \ w_q(Y,1)\ \ ok_q\ r_p(Y)\ \ ret(Y,1)\ \ r_q(X)\ \ ret(X,1)$$

נניח שבמימוש שני ערכי ההחזרה בסוף $ret_p(Y,0)$, וערכי ההתחלה הם אפסים. נטען כי אפשר יהיה לסדר אותה כנדרש באופן הבא:

$$w_p(X,1),\ ok_p,\ r_p(Y),\ ret_p(Y,0),\ w_q...$$

ואכן זהו סידור "הגיוני" שעומד בתנאים שדרשנו. לעומת זאת, אם נרצה $0,0$ לא נוכל למצוא סידור מתאים - צריך לבצע את ה־$write$ לפני ה־$read$ עבור שני המעבדים (לא רק אחד - זאת ראינו כי ניתן לבצע), ואז מבצעת כתיבה שהופכת את הערך של הקריאה. על כן צריך תנאי נוסף:

    * פעולות על אובייקט יראו אטומיות - $\pi|_o$ לגלי (ה־$read$ תמיד מחזיר את ה־$write$ האחרון) - כלומר סידרתי וקונסיסטנטי.

בדוגמא הקודמת עבור $Y=1, X=0$, אם נדרוש גם את תנאי זה, יש לבצע הזזה - הסדרה שכתבנו לא תקפה, אבל עדיין ניתן למצוא אחת כזו.

שימוש באבסטרקציה כזו נותן למשל למפתח סביבת עבודה הרבה יותר נוחה. עם זאת, אנחנו צריכים לעשות את "העבודה השחורה" - המימוש.

3.2.2 מימוש בעזרת bc קוזאלי

הפרוטוקול המוצע עבור מעבד p_i, שלכל אחד יש את העתק מקומי של כל אובייקט X - את ערכו המקומי נסמן ב-$V(X)$:

- בקבלך $W_i(X,v)$:
  - בצע $cbc-send_i\left(\overbrace{w,X,v}^{m}\right)$.
- בקבלת $R_i(X)$:
  - החזר $Ret_i(X,V(X))$
- בקבלך $cbc-rec_i(w,X,v,j)$:
  - $V(X)=v$
  - אם $j=i$, בצע $ok_p(X)$ (כלומר סיום מקומי של פעולת ה-w).

האם מימוש זה עומד בתנאים שביקשנו?

החולשה - מה-$cbc-send$ מקבלים ישר את ה-ret, ולכן בעצם לא מחכים. ה-cbc שומר קונסיסטנטיות פנימית בלבד, ואינו שומר על הדרישה השניה בתנאי הלינאריות. על כן, cbc הוא לא מימוש מוצלח.

3.2.3 מימוש בעזרת סדר מלא - tbc

- בקבלך $W_i(X,v)$:
  - בצע $tbc-send_i\left(\overbrace{w,X,v}^{m}\right)$.
- בקבלת $R_i(X)$:
  - החזר $Ret_i(X,V(X))$.
- בקבלך $tbc-rec_i(w,X,v,j)$:
  - $V(X)=v$.
  - אם $j=i$, בצע $ok_p(X)$ (כלומר סיום מקומי של פעולת ה-w).

עכשיו ה-$writes$ הם בסדר שלם. האם יש צורך בשמירה על סדר שלם גם עבור ה-$reads$?

איך מוצאים את π מתוך σ? נסדר את w ב-$total\ order$, ונשים מיד אחרי w את ה-r המתאים לו. מכאן נקבל סידרתיות מקומית לכל מעבד. לאחר ההודעה של "סיימת" יכולה להתחיל פעולה נוספת. כמו כן נוכל לדעת בדיוק איפה לשים את r. פרמוטציה זו מקיימת את התנאים, ולכן מימוש זה עובד.

אפשר להחליש את הלינאריות ולקבל מימושים אחרים, אך לא נדון בכך.

3.2.4 סימולציה של $Single\,Writer\,Single\,Reader$ מעל $Single\,Writer\,Multiple\,Reader$

אחד הדברים שיכולים לעניין אותנו הוא הדבר הבא:
מעל אובייקט יש single reader, single writer, מעל נרצה לממש שכבה של single writer, multiple readers.

עוד דוגמא - המימוש שלנו דרש ש"כולם ישתפו פעולה". נרחיב את התכונות ונדבר מהן התכונות של σ, π כאשר מעבד נופל. אז אפשר להחליש את הדרישות שלנו למעבדים שהתחילו ועדיין לא הסתיימו. הבעיה במימושים שמקבלים לפעמים היא שיש פעולות שעוד לא הסתיימו - האם דורשים שהמעבדים ישתפו פעולה כדי שהפעולה תסתיים? רמזנו על מימושים שיודעים להתמודד עם נפילות של מעבדים.

תכונה נוספת שמגדירים בתחום הזה היא wait-freeness: נסתכל על סדרה σ, ונניח שיש לה רישא α בו יש פעולה שהתחילה אצל p_i. נרצה שנוכל להשלים פעולה שכזו "בעצמי". זה לא אפשרי במימוש שראינו, אבל קיימים מימושים שכאלו. באופן פורמלי:

הגדרה 3.3 wait-freeness - אם ל-σ יש רישא α שיש בה פעולה שהתחילה אצל p_i, יש השלמה של α ע"י סדרת פעולות של p_i בעצמו שמשלימה את הפעולה.

נראה שיש דברים שאפשר לממש באופן שכזה, ויש דברים שאי אפשר, ונראה מתי זה קורה.

14/1/2012

הנחה - הכתיבה של וקטורים נעשית באופן אטומי!

התכונה הזו על פניו נראית בסדר, אבל כאשר חושבים לעומק יש כאן בעיה, שכן יש תלות בין מה שמעבד עושה לבין מה ששאר המעבדים עושים. כאשר הנושא נלמד בסוף שנות השמונים הוא היה סופר-תיאורטי, אולם בעולם המולטי קור נגזרות של הנושא הזה מצאו את מקומן. למשל, סטודנט של דני הריץ משהו בשני מחשבים, שניהם מולטי-קורים, והתברר שהיו הבדלים משמעותיים בזמן הביצוע - בשל ארכיטקטורה שונה של ה-$cache$. דווקא בזה שהיו יותר רמות של $cache$, הביצועים היו $\frac{1}{10}$ מהמחשב השני. הדברים שנעשה בשבועות הקרובים ישליכו לנושא הזה.

היום בשלבים ננסה לעשות עוד ועוד דבירם שבסופו של דבר הם לינאריים ו-$wait-free$.

אנחנו עוסקים ב-$share-memory$ לא מעל $message\ passing$. מעל השכבה הזו בונים שכבות נוספות שמבצעות אבסטרקציות של פונקציות.

בשבוע שעבר ראינו אבסטרקציה לתא זיכרון, שמישהו מסויים יכול לכתוב בו, ומישהו אחר יכול לקרוא אותו. אפשר גם לדבר על כתיבה\קריאה ביט-ביט, אבל זה לא יעניין אותנו במסגרת הקורס. אם כך היחידה הבסיסית שלנו היא single writer/single reader, ומטרתנו: פיתוח פרוטוקול שידע לספק מעל היחידה הזו single writer multiple reader.

אפשרות פשוטה: בעזרת נעילה, רק אחד יכול לקרוא בו"ז, אבל זו לא המטרה שלנו - נרצה להמנע משימוש במנעולים, ולהרשות ביצוע פעולות בו זמנית. כמו כן, נרצה יכולת למפות לינארית את סדר הפעולות כך שיהיה קונסיסטנטי באופן שראינו בשבוע שעבר.

תזכורת: פעולת ה-$write$ מקבלת ok, פעולת ה-$read$ מקבלת בחזרה את התוכן.

ניתן להוכיח שאי אפשר לפתור את הבעיה הזו, לפי שלפחות אחד ה-$reader$ גם כותב. בפרוטוקול שנפתח ל-$readers$ תהיה גם יכולת כתיבה על מנת לעזור לאחרים ל-$readers$ האחרים לסיים. אנחנו נפתח פרוטוקול בסיסי, ולא אופטימלי.

בניית הפרוטוקול תחילה, ה-$writer$ צריך קאונטר שהוא יקדם בכל פעם שיש כתיבה.

השאלה היא מה עושים עם זה ה-$readers$. ה-$writer$ חייב לכתוב לכל ה-$readers$ (היחידה הבסיסית היא כאמור $1W1R$ ואנחנו פועלים מעליה, כלומר לכל $reader$ יש יחידה שכזה).

- כיצד יתבצע $write$? מעלה את הקאונטר, ואז כותב לכל ה־$readers$ זוג סדור של ערך וקאונטר.

איך אני כ־$reader$ מצליח לקרוא? אני יכול לקרוא את התא המיועד לי, אך אני צריך דרך כדי לדעת מה האחרים קראו לאחרונה ע"מ לדעת האם הערך שלי הכי עדכני. כל $reader$ יהיה $writer$ - הוא יכתוב לכל $reader$ מה הוא קרא (כלומר n^2 תאים).

- כיצד יבצע $read$? קורא את הערך עבור עצמו, קורא את הערך המיועד לו אצל האחרים, לוקח את הערך המקסימלי - סוג סדור של ערך וקאונטר, מעדכן את ה־n־יה שלו עבור ה־$readers$ האחרים, ומחזיר את הערך הזה.

כיצד נסדר את הפעולות? נסדר את פעולות ה־$write$ על ציר הזמן לפי ה־ok שלהם.

לגבי ה־$readers$ - יש לנו סדרה ארוכה של התחלה וסיום, ונרצה להשליך על בסיס הזמן שבנינו עבור ה־$writes$. נסתכל על התשובות ש־$read$ מחזיר (כלומר על נקודת הסיום של הפעולה, לא על ההתחלה), מתחילים עם התשובה הראשונה, ונשים אותו מיידית לפני ה־$write$ הבא (קריאה נוספת עם אותו הערך "תדחף" בין ה־$write$ לבין ה־$read$ הקודם). סדר שכזה דואג שאם $read$ הסתיים לפני שאחר התחיל, הסדר נשמר - אם $read$ מסויים הסתיים לפני שאני התחלתי, אין שום דרך שאהיה מסודר לפניו. מכך קיבלנו את הלינאריות שאנו מחפשים.

3.2.5 סימולציה של $Single\ Writer\ Multiple\ Reader$ מעל $Multiple\ Writes\ Multiple\ Reader$

נמצא פרוטוקול דומה, ושוב לא נשתמש במנעולים.

כאן הבעיה היא בסידור ה־$writers$ - כבר ראינו בעיה דומה בקורס, על כן נטפל בה באופן דומה.

בניית הפרוטוקול מספר קוראים - n, מספר כותבים - m, $n \geq m$.

כל $writer$ יסתכל על האחרים כדי לשייך לעצמו $timestamp$. השאלה היא האם צריך את הכל, או רק את הערך המקסימלי? וקטור $timestamp$ הוא מושג שכבר ראינו בקורס, ומהווה את ה־n־יה המופיעה אצל האחרים. כשהשתמשנו בכלי זה, המקום היחידי שעניין אותנו (בקואזליות) שמשהו ממש קודם למשהו אחר, כאשר אין יחס סדר מלא לא היה אכפת לנו, שכן הם "כאילו" פעלו בו זמנית. כאן עם זאת חשוב לנו לדעת איך לשים את ה־$writers$ כי אחרת לא נדע למקם נכון את ה־$readers$. כלומר, צריך למצוא איכשהוא לממש סדר מלא.

יש לנו $writer$ אחד שאסף לעצמו n־יה, $writer$ אחר שאסף לעצמו n־יה נוספת, ויש בהם ערכים זהים וערכים שונים (אולי היו כתיבות בין לבין). נביט ב־$entry$ הראשון בו וקטור אחד שונה מהשני. איך נקבע מי ראשון? מי שהערך שלו קטן יותר - אז בטוח שהקטן התחיל לפני הגדול. יכול להיות שבאחרים יהיו ערכים סותרים, שכן תלוי מתי כל אחד דגם את האחרים, ועל כן מסתכלים על הראשון מביניהם ששונה (אם ה־$readers$ ישתמשו באותו כלל, לא תהיה לנו בעיה - נראה זאת בהמשך).

אם כך מהו הערך שמופיע בכל אחת מ־$entries$ בוקטור ה־$timestamp$? m־יה! איזה מהם יעניין אותנו? ל־k יש ערך פנימי שהוא מקדם, ואותו צריך להחזיר. ובאופן מסודר:

עבור תהליך p_k:

- בקבלך $write_{MWMR}(v)$:

$$\forall 1 \leq i \leq m: \ t[i] := T_{1WMR}[i].i \quad (1)$$
$$t[k] = t[k] + 1 \quad (2)$$
$$V_{1WMR}[k] := \langle v, t \rangle \quad (3)$$
$$T_{1WMR}[k] := t \quad (4)$$
$$return\ ok \quad (5)$$

- בקבלך $read_{MWMR}$: סורק את הוקטור V_{1WMR}, מחפש את t המקסימלי (לפי הסדר שהגדרנו), ומחזיר את הערך המתאים לו.

ההבדל בין האלגוריתם הזה לבין האלגוריתם הקודם - ה־$readers$ לא צריכים כאן לכתוב **מעל**, שכן את עבודת התחזוקה כבר מבצעים ה־$writers$ (המימוד של $1WMR$ מבצע את הכתיבה).

כיצד נסדר את הפעולות? נזכור כי על וקטור ה־$timestamps$ יש לנו סדר מלא.
נטען כי אם $reader_j$ התחיל אחרי ש־$reader_i$ סיים. מדוע? מכיוון והמימושים שלנו ל־$1WMR$ שומרים לינאריות, כאשר j יתחיל לבצע את ה־$read$ שלו כדי לקבל את ה־$timestamp$ שלו, הוא יקרא ערך ממש גדול מזה של i. לכן הסדר המלא שהגדרנו שומר לנו על הלינאריות.
בהנתן הטענה מעלה, נסדר:
מכיוון והוקטור הזה נותן לנו סדר מלא, נסדר את ה $writers$ לפי סדר זה.
את ה־$readers$ נסדר באופן הבא: כל $reader$ מחזיר ערך שקשור לאיזשהוא $timestamp$. נסדר אותם כמו מקודם, אולם לפי t ולא לפי הקאונטר: כל מי שהחזיר את אותו הערך יהיה באותו סגמנט, והסדר הפנימי יהיה אותו דבר (שוב לפי t). אם $reads$ באותו סגמנט הסדר נשמא מהטענה מעלה, ואם קוראים ערכים אחרים, יהיו בסגמנט שונה, ועל כן הסדר ישמר (מי שקרא את ערך מאוחר יותר יסווג לסגמנט הבא).

3.2.6 אבסטרקציה של פעולות $scan$ ו־$update$

עד עתה היה רק תא זיכרון אחד. כעת נניח ולכל תהליך יש איזור\תא זיכרון משלו, ואיזור משותף לכולם. נרצה למצוא מקביל ל־$snapshot$ אטומי של ה־$shared\ memory$, כך שפעולות ה־$scan$ וה־$update$ יהיו מסודרים.
זה לא פשוט - אחד עובד יותר מהר, אחר לאט, והערכים כל הזמן משתנים. בכלל לא ברור שאפשר לעשות זאת, אבל אפשר!
הקושי הוא שבכל פעם שאנחנו עוברים על הזיכרון, הם יכולים להשתנות. אם עברנו פעמיים ואף ערך לא השתנה נדע שהצלחנו, אבל יתכן וזה לא יקרה אף פעם.
נסתכל על אותו $updater$ שפעל בזמן שעשינו $scan$. נצטרך שהוא יעזור ל־$scanners$! ה־$updater$ יוציא $snapshot$, כלומר יעשה $scan$, ויכתוב ערך ואת ה־$snapshot$ הזה. אבל אם הרבה עושים $scan$ (גם $updaters$ וגם $scans$), אולי כל אחד מפריע לאחר... ב־$timestamp$ הסתפקנו בלהסתכל על $entry$ יחיד ששונה, אבל כאן ה־$scan$ הוא וקטור שאת כולו צריך להחזיר, **ואסור** שיהיה בו ערכים סותרים.

האם יש כאן $livelock$**, כלומר, האם יש** $updater$ **המנסה לעשות** $scan$ **ולא מצליח?** אם הכותב הצליח לכתוב, אז הוא יצליח לעשות פעמיים סריקה "נקיה", אחרת לא יצליח לבצע כתיבה. כל אלה שמנסים לכתוב בו"ז לא יצליחו עד שמישהו יצליח לעשות פעמיים $scan$. כלומר אין $livelock$, רק צריך לדעת איך לסמן.

איך ה־$scanner$ **וה־**$updater$ **יתקשרו?** נרצה שהם יצליחו לסמן אחד לשני אם מאז שהתחיל לעשות $scan$ נעשה $update$. נגדיר תא זיכרון שהוא "דגל" לכל זוג. לכל אחד גם יש משתנה מקומי. איך הדגל יסמן אם ה־$updater_j$ התחיל אחרי שה־$scanner_i$ הדליק את הדגל? ה־$scanner$ ידליק כשהוא מתחיל, $updater$ יחכה כשהוא נכנס, ואז $scanner$ יוכל לבדוק ולראות את הקריאה שלו הופרעה או לא.

הבחנה בין שני $writers$ יש להבחין בשינויים בין $writers$ רציפים. נצטרך איזשהוא דגל נוסף שכל $writer$ יעלה וירד.

איך יתבצע $scan$**?** אם עשינו שתי סריקות והן זהות, הצלחנו. אחרת, נצטרך לתפוס את זה ולקבל את התמונה הנכונה. המנגנון שהגדרנו עוזר אבל לא מספיק ־ אנו יודעים אם הסריקה לא רלוונטית, אבל אנחנו לא יודעים את הסדר ־ אולי ה־$updater$ סיים לפנינו, אולי אחרינו... כשאנחנו רואים את הביט המכובה, אנחנו לא יודעים שמשהו כבר שונה.

צריך למצוא $updater$ שהתחיל את ה־$scan$ שלו אחרי שאנחנו התחלנו. איך נדע את זה? אם מתבצעות שתי כתיבות ־ אחרי הכתיבה השניה אנחנו יודעים ש"אנחנו בסדר" (בדומה לסנכרון שעון), שכן היא בטוח התחילה אחרי ה־$scan$ שלנו. כלומר, עלינו לדעת כמה פעמים נעשה $update$. אפשר לספור ולקחת את הפעם השניה.

המנגנון ישתמש במנגנונים האלה לזהות שינויים ־ אם $updater$ מסויים ביצע שני שינויים או יותר שראינו ־ כלומר פעמיים כיבה את הביט, אז נהיה "במצב טוב" (היה לו $write$ שהתחיל אחרי שאנחנו התחלנו) ־ וצריך לקחת את ה־$scan$ שלו. אבל יכול להיות שהוא כיבה את הביט ולא הספיק עדיין לכתוב. לכן, אם כיבה 3 פעמיים, "אני מסודר" ־ בטוח שהשינוי השני כבר התרחש.

הפרוטוקול

- $scan$:
  - אתחל וקטור שיציין כמה שינויים כל אחד עשה.
  - הדלק דגלים.
  - אוסף זכרונות.
  - איסוף שני.
    בדוק : אם האוספים זהים ־ גמרת.
  - אם "יש שינוי" (הדגל או הזיכרון השתנה), זכור את עושה השינוי (הוסף 1 לוקטור במקום הרלוונטי), והגדל את הקאונטר הפנימי שלך.
  - אם יש מישהו ששינה 3 פעמים, החזר את ה־$scan$ שלו. אחרת, חזור לשלב "הדלק דגלים".
- כותב־$updater$:
  - $scan$.

- כתוב (v, S, b).
- כבה דגלים.
- החזר ok.

ב־$scan$ אפשר לעשות את הלולאה $2m+1$ פעמים, אך לא יותר, ולכן אין $livelock$. זה לא יעיל, אבל מכאן נגזרו תובנות ששינו את החומרה.

כיצד נסדר את הפעולות על ציר הזמן?

תחילה, נסתכל על ה־$scanner$ המוצלחים במובן שקיבלו שני אוספים זהים, ונסדר אותם.
לכל $scan$ אחר שהצליח, הוא הצליח כי "הציץ" אצל מישהו אחר, אנחנו יודעים מיהו. אזי נסדר את ה־$updaters$ הנותרים לפי הסדר ה־$scan$ שלהם (למציאת ה"חלון") ואז לפי המקום שבו הם כתבו את הערך.
לבסוף נסדר את ה־$scans$ הנותרים אחרי ה־$updater$ המתאים להם (הם ראו את ה־$scan$ אחרי שה־$updater$ המתאים כתב אותו לזיכרון).

21/1/2012

בשבוע הבא נקשור את המשך מסע ה־$wait-freeness$: מה אם נרצה לקרוא ולכתוב בצורה אטומית? זה אין לנו עד עכשיו, ועם זה נתמודד בשבוע הבא.
היום נסיים את הנשוא בו עסקנו בשני השיעורים האחרונים.

בעיה נוספת נתחיל בבעיה הבאה: ישנם שלושה תאי זיכרון המאותחלים לערך $\perp$, ושני מעבדים ־ p, q (נגיד שכותבים את הערכים $1, 2$ בהתאמה), כל אחד יכול לכתוב לשניים מהתאים הללו ולקרוא את שלושתם, דהיינו, ישנו תא משותף לשניהם. האם אפשר להגיע להסכמה, בהנתן הפעולות שהגדרנו בשיעור הקודם, שהוא $wait-free$?
כן! מה קורה ל־p כאשר הוא רץ לבד? מתחיל לרוץ, רואה $\perp$ בכל התאים. הוא כותב את שני התאים שיכול, ורואה $\perp$ בתא השלישי, ואז יודע שניצח. אם הוא רואה ערך שאינו $\perp$ בתא השלישי, הוא יודע שהוא "לא לבד" ־ האמצע יקבע מי ניצח ־ המעבד יעבור לתא האמצעי, ויבדוק מה כתוב שם.
כאן אנחנו "קרובים מאוד", אולם חסר לנו נדבך נוסף.
כיוון אחד ־ "כתיבה לשיעורין" (עוד על כך בהמשך).
כיוון שני ־ ננסה להמשיך את הרעיון של כתיבה אטומית. חסר לנו המקרה בו p סיים את הפעולה לפני ש־q בכלל התעורר. אנחנו צריכים לדאוג שכאשר q מתעורר אחרי p, q יוכרז כמפסיד, לא מנצח. בגלל הכתיבה האטומית, אנו יודעים שבשלב הבדיקה בשלישי לא יהיה $\perp$. אז, אם באמצע מופיע הערך שלו נקבע כהפסד, כדי להתמודד עם המקרה הזה.
הפרוטוקול המסודר:

- מנסים לכתוב אטומית לשני תאי הכתיבה.
- בודקים את השלישי ־ נסמן את ערכו ב־v':
  - אם $v' = \perp$ ניצחת.
  - אחרת, קרא את הערך בתא האמצעי v:
    - אם $v' = v$, ניצחת.
    - אם $v' \neq v$, בחר v'.

האם אפשר להחליט על שני הערכים שונים? הכלל מוגדר היטב - הערכים בכל התאים כתובים, ולכן אין הסכמה על שני ערכים שונים.

אם כך לא היינו צריכים כתיבה לשיעורין, ועבור שני מעבדים הבעיה פתירה.

האם ניתן לפתור ללא כתיבה אטומית לשני תאי זיכרון? נשים לב כי $scan$ מאפשר קריאה אטומית של התאים, כך שאין אנו מאבדים את היכולת הזו. כמו כן, נשים לב כי בשבוע שעבר כתבנו לתא אחד, ולכן הבעיה הקודמת בה השתמשנו בכתיבה אטומית לא כוסתה בדיון בשבוע שעבר.

הערה נוספת: אין לנו כאן ריצה אינסופית, שכן הפתרון שאנו מחפשים הוא $wait-free$.

ננתח את האפשרויות: אם אין כתיבה אטומית, הפרוטוקול הקודם לא עובד - יכול להיות ששניהם מתחילים לרוץ, אחד כותב, חושב שהוא מפסיד, והשני מגיע וחושב שהוא מפסיד גם.

במודל הנוכחי FLP אפשר לעשות סימולציה של shared memory (לא ראינו זאת), אז FLP קיים, לכן אולי מסתתר כאן FLP, אבל צריך להבין מהי המשמעות שלו במסגרת המודל הזה. הסביבה אסינכרונית, יש פעולות $read/write$. אז בעצם, אם נסתכל על ה-$flow$ של הפרוטוקול, תמיד נקבל עץ, ועץ האפשרויות הוא כל האפשרויות לשזור את כל ה-$read$-ים וה-$write$-ים.

המצב ההתחלתי בעץ "בלשון FLP" - יש מצב דו ערכי התחלתי, בו הם התחילו עם ערכים שונים. אם יש FLP אז הם צריכים להגיע להחלטה. כל אחד מהעלים הוא חד ערכי, כי בכל מסלול בסופו של דבר צריך להחליט, ומעבד אם רץ בעצמו בעץ בסופו של דבר מחליט. לכן, בעץ יש כל מיני קודקודים שהם "קודקודים קריטיים" - מעבירים בין מצב דו-ערכי לחד-ערכי. נסתכל על קודקוד קריטי כזה - יש מתחתיו עלים עם 0, ויש מתחתיו עלים עם 1. מה קורה בקודקוד כזה? נעבור על כל האפשרויות, כאשר נסקור שתי השתלשלויות מהעץ - מהאחד מגיעים לעלה החלטה על 0, והאחר על 1:

- נניח שגם p וגם q עושים פעולת $read$. אז אחרי R_p אפשר לבצע את R_q, וגם אחרי R_q אפשר לבצע את R_p, שכן קריאות לא משפיעות על מצב המערכת - קיבלנו משהו דומה ללמת המעויין - המצבים זהים מבחינת המעבדים! אבל זו בסתירה לעובדה שבאחת האפשרויות הם מחליטים על 1, ובאחרת על 0, לכן סוג כזה לא אפשרי במצב קריטי.
- נניח ש-p מבצע קריאה ו-q מבצע פעולת כתיבה. q חייב להשאר באותו מצב: העובדה ש-p עשה קריאה לא משפיעה עליו כהוא זה. אבל, באחד המצבים על q להחליט 1, ובאחר 0 - בסתירה.
- על כן נותר המקרה בו שניהם עשו $write$:
  - אם עשו $write$ לשני תאים שונים, אז שני המצבים עד עכשיו שונים. אבל אחרי הכתיבה הראשונה תתבצע הכתיבה השניה (זו שהתרחשה קודם בעץ השני), לכן כל ההמשכים משני העצים שקולים, בסתירה.
  - אם עשו $write$ לאותו התא, אז ב-$branch$ אחד p כתוב, השני q כותב ואז p כתוב, ואז p לא יראה הבדל מאותה נקודה, ושוב נקבל סתירה.

אם כך, בקריאה וכתיבה לא ניתן לפתור קונצנזוס.
על כן, יש צורך ב"קפיצה" - יכולת לכתוב ולקרוא יותר מתא בודד בבת אחת. כאן אנו מקבלים הוכחה לכך שיש דברים שלא ניתן לעשות רק עם רגיסטרים של $read$ ו־$write$.

Fetch and Set בקורס מערכות הפעלה הכרנו את המנגנון (בשם test and set):

Fetch and Set

$$faS(v)$$
$$prior = value$$
$$value := v$$
$$return\,(prior)$$

נראה בשעה הבאה איך אלמנט התוכנה הזה יכול לעשות את מה שלא הצלחנו בעזרת הכלים שלנו עד כה.
האם אפשר להגיע לקונצנזוס על שני מעבדים בעזרת מנגנון זה?
נטען שכן, בעזרת תא קריאה\כתיבה אחד. נניח כי הערך ההתחלתי של התא $\perp$ שונה מהערכים של המעבדים.

הפרוטוקול p, q יקראו ויסמנו - $v' := fa\ s(v)$. אם $v' = \perp$, החלט על ערכך v, אחרת החלט על v'.
הפרוטוקול הזה כמו שהוא עובד עבור שני מעבדים, אך לא באותה המתכונת עבור מספר גדול יותר של מעבדים.

3.2.7 RMW

המנגנון שראינו למעלה הוא חלק מסדרה של מנגנונים בשם RMW - Read Modify Write הפועלים באופן כללי באופן הבא:

$$prior := value$$
$$value := f(value, v)$$
$$return\ f'(prior, value, v)$$

למשל:

- Fetch & Increment (v)
$$f(value) := \quad value + 1$$

- Fetch & Add (v)
$$f(value) := \quad value + v$$

•

Conditional Set (v, p')

$$f: \quad if\ value := p',\ then\ value := v$$

יש הרבה פונקציות כאלה שאנשים חשבו עליהם על השנים. המחשבים הראשונים שפותחו היה בהם אלמנט של fetch & set. בשעה זו נראה מדוע בגלל דברים שפותחו בתיאוריה שינו את הפועלה הזו, לאיזו פעולה נראה בהמשך.

מכאן יש כמה אפשרויות להתקדם. אם נסתכל על כל הפונקציות האפשריות, אפשר לחלק אותן לכמה משפחות:

- פונקציה קומוטטיבית - כלומר פונקציה המקיימת:

$$f_p(f_q(value)) = f_q(f_p(value))$$

למשל fetch&add היא פונקציה קומוטטיבית.

- f הוא set - "הערך מכסה את עצמו", כלומר פונקציה המקיימת:

$$f_p(f_q(value)) = f_p(value)$$

fetch&set היא פונקציה שכזו.

- פונקציה שלא מקיימות את אחד התנאים הללו - למשל Conditional Set.

האם עם פונקציה כזו ניתן לפתור את הקונצנזוס עבור יותר משני מעבדים?

עם Condition set: וריאנט על הפרוטוקול הקודם - בצע $v' := \text{Condition Set}(v, \perp)$, אם $v' = \perp$ הכנס את הערך שלך, אחרת החלט על הערך v'[6].

הערה 3.4 ניתן להראות שאם יש Conditional Set אפשר לבצע סימולציה של כל אופרטור על זיכרון משותף שאפשר לחשוב עליו, והפתרון הזה הוא ב-$wait-free$.

למעשה ה-Condition Set הוא משפחה שלמה, ואפשר לעשות איתו הרבה מאוד דברים מרתקים, כלומר זוהי פונקציה מאוד חזקה. האם אפשר לפתור את הבעיה שלנו בעזרת פונקציה אחרת?

עם Fetch&Set: נניח שיש שלושה מעבדים - p, q, r, ויש לנו מספר לא מוגבל של תאי זיכרון.

פרוטוקול מוצע: לכל אחד יש תא זיכרון פרטי, שרק אחד מהם יכול לכתוב אליו. בנוסף לכך, לכל זוג יש תא שאליו אפשר לעשות Fetch&Set: pq, qr, rp.

- עושים Fetch&Set על pq, ולכן בתא הזה רשום איזה ערך בו, וכל אחד שניגש יכול לדעת איזה ערך רשום שם).
- עושים Fetch&Set על qr, ומנסים לכתוב את הערך מ-pq לתא זה, וכך במעגל.
- מעבד מחליט על הערך שהגיע אליו בפעם השניה שביצע את הפעולה.

אבל, פרוטוקול זה לא עובד...

נחזור לעץ ה-FLP, נראה מה עובד ומה לא עובד שם, ולפי כך ננסה להגיע לפרוטוקול.

[6] "יש כמה דברים מאלפים בתחום הזה", לפי המרצה.

נתח שוב קודקוד קריטי בעץ ההחלטות של פרוטוקול נניח לשם פישוט כי הערך הוא בינארי (ברור כי אם מוכיחים לערך בינארי אפשר להכליל). כנראה שהקריטיות היא ה־f_{set} אותה מפעילים בצומת זה. יש אינפורמציה ל־q, p, אבל אם בצומת הקריטית מפעילים את f על זוג, השלישי לא יראה הבדל בין השניים, כי ההבדל הוא באינפורמציה פנימית שהוא לא יודע עליה ־ מכיוון ו־r צריך להחליט ולא רואה הבדל, נקבל סתירה. זה נכון לכל פונקציה שעושה set.

מה לגבי פונקציה קומוטטיבית? p, q ידעו מי ניצח ביניהם, r שוב לא יהיה מסוגל להחליט. כלומר, אפילו שלושה מעבדים לא ניתן להגיע להחלטה.

על כן, לשתי המשפחות הללו יש מגבלה אינהרנטית, בעוד שלמשפחה השלישית אין מגבלה זו ־ התנאי מבטיח החלטה שכולם יראו אותה.

הגדרה 3.5 $\#\ Consensus$ של אופרטור הוא לכמה מעבדים האופרטור יכול לפתור קונצנזוס.

למשל $\#\ Consensus$ של Conditional Set הוא ∞, של Fetch&Set הוא 2.

קוריוז אינטלקטואלי: ניתן להוכיח שאם נותנים פעולה אטומית על k תאי זיכרון, אפשר לפתור את הבעיה עבור $2k - 2$ תהליכים.

ב־$Shared\ Memory$ יש עוד עולם ומלואו. אם היה עוד שיעור היינו רואים את הסימולציה של כל אופרטור בעזרת Conditional Set.

הרבה מהדברים האלו ניתן לעשות ב־$Self\ Stabilization$ אבל לא את כולם. לא חקרו את הנושא הזה לעומק, כי "מה שמעניין" הוא ב־ST וטעויות.

בשארית הזמן שנותר לנו ־ נביא פתרון אפשרי לתרגיל 3:

פתרון אפשרי לתרגיל 3

בתרגיל היינו צריכים להגיע ל־VAC. היתה לנו את פונקציית $check\,(v)$, כדי לוודא שהערך הוא אפשרות להסכמה. כמו כן יש אלגוריתם $LEAD$, כשבהסתברות p מקבלים מנהיג טוב. מכיוון ומדובר באלגוריתם אסינכרוני, אפשר תמיד לחכות ל־$n-t$ טובים שישלימו "איתי" את המהלך הנוכחי.
הפתרון שנציג הוא לא הכי טוב שאפשר, אלא המטרה היא להראות גישה קונספטואלית ואינטואיטיבית.
ננסה להשתמש ב־GC שראינו , כמודול בו ניתן להשתמש ל־$valid\ send\,(p,m)$ - שולחים בו את ההודעה, אם קיבלתי הודעה ממישהו, עושים echo לכולם, ומחכים עד שמקבלים $n-t$ על ההודעה הזו. אם קיבלתי $n-2t$ (בעצם אפילו מספיק רוב) ערכים זהים, זה ערך השליחה (אם היו חתימות, אפשר היה לחכות ל־$n-t$).
כלומר, משתמשים בכלי הזה למשלוח כל הודעה. אם לא מתקבלות מספר ההודעות, מתנהגים כאילו ההודעה לא נשלחה כלל. זה מבטיח שאם שני טובים קיבלו ערך, הם קיבלו את אותו הערך.

- שלב 0: שלח לכולם ערכך (דרך $valid\ send$), ובדוק $check\,(v)$. לכן בהמשך, כל ערך שהוא ערך אפשרי להחלטה יהיה מאומת. את שלב זה נריץ ברגע כל הזמן, ונשמור את ערכי המעבדים שקיבלנו כל הזמן.
- הפרוטוקול עצמו הוא לופ של כמה שלבים - בסבב i:
  - $(i,1)$: שלח ערכך m לכולם.
  - $(i,2)$: אסוף $n-t$ הודעות $(i,1)$. אם יש $n-2t$ זהות עם ערך v, שנה m להיות v. אחרת, $m=\perp$.
  - $(i,3)$: שלח m לכולם.
  - $(i,4)$: אסוף $n-t$ הודעות $(i,3)$.
    - אם יש v שהתקבל לפחות $n-3t$, קבע m להיות v.
    - אם יש v שהתקבל לפחות $n-2t$, החלט v, אחרת $m=\perp$.
    - אחרת...
  - $(i,5)$: שלח $(i,4)$ לכולם.
  - $(i,6)$: מחכים ל־$n-t$ ערכים של $(i,4)$.
  - $(i,7)$: הרץ $LEAD$ - בחר מנהיג טוב בהסתברות p. לפחות חצי מהטובים השתתפו ב־$(i,3)$, לכן ישנה הסתברות קבועה $p'=\frac{p}{2}$ שבחרנו טוב, והוא שלח ב־$(i,3)$ משהו.
    אם החלטתי, המנהיג לא מעניין. אם לא החלטתי, אקח את הערך שהמנהיג שלח ב־$(i,4)$, וזה יהיה הערך לסיבוב הבא. אם ערך זה הוא $\perp$, אבחר את הערך שהמנהיג שלח "בהתחלה" (שבשלב 0).

מה קיבלנו?
נניח שקיבלנו ב־$(i,7)$ מנהיג טוב. אם יש טוב שהחליט ב־$(i,4)$, לכל הטובים יש את אותו הערך - קיבלו לפחות $n-2t$, לכן הם שלחו את הערך הזה, ולכן הטוב שלח אותו. אם אף אחד לא החליט ב־$(i,4)$, אבל שלח ערך ב־$(i,5)$, אם יש מנהיג טוב, יש לכולם את אותו הערך.
לכן, בהסתברות שהמנהיג הוא טוב ושלח ערך, "אנחנו מסודרים". אם המנהיג טוב ושלח $\perp$, יש שני מקרים: אם מישהו החליט, המנהיג לא יכל לשלוח $\perp$, לכן במקרה זה חלק קיבלו וחלק לא, ואף אחד לא החליט, ולכן כולם יקבלו למנהיג ב־$(i,7)$.
מה יקרה בסוף הסבב הבא? כולם יחליטו בסופו? לכן יש הסתברות קבועה לכך שכולם יקבלו את אותו הערך, ובסבב לאחר מכן יחליטו עליו.
כמה פינות להשלים: נרצה להגיד שאם המנהיג שלח $\perp$, יש ערך בשלב 0 עליו נוכל להחליט. אולם, יכול להיות שהודעות שלב 0 שנשלחו עדיין לא הגיעו אלי. לכן יש צורך בחיזוק של שלב זה: יש לחכות ל־$n-t$ שיגידו שסיימו את שלב 0. בכך מובטח שאם המנהיג הוא טוב, "מספיק" טובים סיימו.
למעשה, יש דרך יותר פשוטה לפתור את זה: המנהיג יכול, כששולח $\perp$, שולח את הערך