סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

הרצאה 1– מבוא ומודלים של מערכת

מה זה מערכת מבוזרת?

מערכת מבוזרת זה אוסף של מכשירים בעלי יכולת חישובית שיכולים לתקשר זה עם זה.
לכל מעבד במערכת מבוזרת יש בדרך כלל אג'נדה אישית, כולל שיתוף במשאבים, זמינות וסובלנות לתקלות, המעבדים צרכים לתאם את פעולותיהם.

למה מערכות מבוזרות חשובות?

מערכות מבוזרות נמצאות בכל מקום! האינטרנט, מכשירי טלפון וכדומה.
מערכות מבוזרות מאפשרות לנו:
– לחלוק מידע בין גופים, לטפל במידע במימדים גדולים יותר, חישוב מקבילי על גבי מכשירים רבים, לבנות מערכות הנפרשות על שטח רחב, לבנות תשתיות תקשורת, לבנות מערכות חזקות וסובלניות לתקלות.

למה ישנן סוגים שונים של מערכות מבוזרות?

במערכות מבוזרות, עלינו לטפל בהמון היבטים ואתגרים מלבד אלה שנמצאים בבמערכות הלא מבוזרות.
חלק מהאתגרים במערכות מבוזרות הם:
– איך לארגן מערכת מבוזרת, איך לחלוק מידע/חישובים, תשתיות תקשורת.
– לעיתים קרובות אין זמן גלובלי.
– תיאום בין צמתים מרובים.
– הסכמה על צעדים לביצוע.
– בנוסף התמודדות עם מערכת אסינכרונית שעלולה לאבד הודעות, צמתים לקויים, עצלניים, זדוניים או אנוכיים.

למה צריך תיאוריה?

במערכות מבוזרות אין לנו את סוג הכלים לניהול סיבוכיות כמו בתוכנות רציפות רגילות.
הסיבה העיקרית לכך היא: הרבה עיכובים בלתי צפויים, כישלונות, פעולות, בלתי צפויים. לאף צומת אין תצוגה מרחבית. מה שמוביל להמון חוסר ודאות.
הרבה יותר קשה ליצור מערכת מבוזרת תקינה: חשוב שיהיו כלים תיאורטיים להוכיח נכונות. הנכונות עשויה להיות תיאורטית, אך למערכת שגויה יש השפעה מעשית.

סוגי מערכות מבוזרות

ישנם שני מודלים אבסטרקטיים ללימוד מערכות מבוזרות:
1. העברת הודעות: הצמתים מתקשרים על ידי העברת הודעות. טופולוגיה מחוברת לחלוטין או רשת שרירותית.
2. חלוקת זיכרון: התהליכים מתקשרים על ידי קריאה/כתיבה מ/אל זיכרון גלובלי משותף.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

העברת הודעות

משמש כמודל עבור מערכות ורשתות גדולות. למעט מערכות בקנה מידה קטן, מערכות אמיתיות מיושמות על בסיס החלפת הודעות.

זיכרון משותף

מודל קלאסי עם בעיות תיאום רבות. מדגים את אופן הפעולה של מעבדים מרובי ליבות וגם תוכניות מרובות תהליכים במחשב יחיד. הפשטה נוחה ביותר לתכנות.

זיכרון משותף אל מול העברת הודעות

באופן כללי, שני הדגמים יכולים לדמות זה את זה, כלומר ניתן ליישם את הפונקציונליות של מערכת זיכרון משותפת על בסיס החלפת הודעות. וניתן ליישם את הפונקציונליות של מערכת העברת הודעות על בסיס שימוש בזיכרון משותף.

- רוב הדברים שנדון בהם קיימים בשני המודלים.
- נלמד את שני המודלים.

סנכרון

מערכות מסונכרנות: המערכת פועלת בצעדי זמן סינכרוניים (בדרך כלל נקראים סיבובים).
סיבוב r מתרחש בין הזמן r−1 לזמן r.

- העברת הודעות מסונכרנת: בסיבוב r−1, כל תהליכים שולחים הודעות. ההודעות מועברות ומעובדות בזמן r.
- חלוקת זיכרון מסונכרנת: בכל סיבוב, כל תהליך יכול לגשת לתא אחד בזיכרון.
- מערכות אסינכרוניות: מהירות התהליכים ועיכובים בהודעות הם סופיים אבל אינם ניתנים לחיזוי.
- הנחה: מהירות התהליך / עיכוב ההודעות נקבעים על ידי המסלול הגרוע ביותר על ידי מתזמן יריב.
- העברת הודעות אסינכרונית: ההודעות תמיד מגיעות (בביצועים ללא כישלון). זמן העיכוב של ההודעות הוא שרירותי.
- זיכרון משותף אסינכרוני: כל התהליכים בסופו של דבר מבצעים את הצעדים הבאים שלהם (בביצועים ללא כישלון). מהירויות התהליכים שרירותיות.
- ישנן מודלים חישובים בין מערכות סינכרוניות לחלוטין לאסינכרוניות לחלוטין.
- גבול של עיכוב הודעות או מהירות של תהליכים: צמתים יכולים למדוד הפרשי זמן ויש גבול עליון של עיכובים/ זמן שלוח לביצוע הודעה.

- סנכרון חלקי: יש גבול עליון על המהירות של התהליכים ועיכוב של ההודעות.
  גרסה 1: המגבלה אינה ידוע לצמתים או לתהליכים.
  גרסה 2: הגבול העליון מתחיל להשפיע בזמן לא ידוע.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Failures– כשלים

- כישלון קריסה: צומת או תהליך מפסיק לפעול בזמן הריצה. יכול לקרות במהלך סיבוב, כאשר רק חלק מההודעות שודרו.
- כישלון ביזנטי: צומת או תהליך מתחיל להתנהג באופן שרירותי לגמרי.
- כישלון השמטה, Omission Failure: צומת תהליך או ערוץ תקשורת מפסיק לעבוד באופן זמני. לדוגמה, חלק מההודעות אבדו.
- חוסן, Resilience: מספר צמתים או תהליכים כושלים.

נכונות של מערכת מבוזרת

כאשר אנו מתעסקים עם מערכות מבוזרות ופרוטוקולים, יש לנו עקרונות שונים לנכונות.
שלושת החשובים ביותר הם:

- בטיחות, safety: שום דבר רע לא קורה אף פעם.
- חייה, liveness: בסופו של דבר קורה משהו טוב.
- הוגנות, fairness: בסופו של דבר משהו טוב קורה לכולם.

בטיחות

שווה ערך לכך שאין מצבים רעים להגיע אליהם.
לדוגמה: צומת דרכים בכביש, בכל נקודת זמן, לכל היותר אחד מהכיווני נסיעה יהיה ירוק.
הוכחת בטיחות: לעיתים קרובות נוכיח את הבטיחות כך שנראה שכל מעבר ממצב לממצב אפשרי כלשהו שומר על מערכת בטוחה.

חייה

לדוגמה: המייל שלי מועבר אל היעד.
הערה: לא מדובר על חלק ממצבי המערכת אלא ביצוע כל המערכת. על החלק להיות בזמן מוגדר.
הוכחת חיות: ההוכחות בדרך כלל תלויות בערכי חיות אחרים. למשל, כל ההודעות במערכת מועברות בסופו של דבר.

הוגנות

נגרר מחיות, בתוספת מניעת הרעבה.
הרעבה starvation: צומת או תהליך שלא יכול להתקדם.
דוגמה: מערכת שמספקת אוכל לאנשים.
חיות: מישהו מקבל אוכל.
הוגנות: המערכת מספקת מספיק אוכל לכולם.

בטיחות חיות והוגנות

נחזור לדוגמה של הרמזורים.
בטיחות: לכל היותר אחד מהאורות ירוק בכל נקודת זמן.
חיות: יש אור ירוק לעיתים קרובות.
הוגנות: בכל הרמזורים יש אור ירוק לעיתים קרובות.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

העברת הודעות–בצורה יותר פורמלית

באופן כללי ננסה לשמור על פורמליזם נמוך ככל האפשר אך על מנת שנוכל להתווכח על נכונות יש צורך בפורמליזם כלשהו.

מודל בסיסי למערכת:

- מערכת מורכבת מ–n צמתים דטרמיניסטים $v_1, \ldots\ldots, v_n$ תהליכים ומערוצי תקשורת.
  - ההנחה היא שהצמתים ממוספרים $1, 2, \ldots, n$ כאשר n ידוע.
  - לעיתים נרצה להרפות מתנאי זה.
- בכל נקודת זמן לכל צומת v_i קיים מצב פנימי Q_i.

מודל מבוסס אירועים

מצב פנימי של צומת: קלט, משתנים מקומיים, אולי כמה שעונים מקומיים. היסטוריה של כל רצף האירועים שנצפו.

סוג האירועים:

אירוע שליחה: צומת v_i שם הודעה על ערוץ התקשורת עבור צומת v_j כלשהו.

אירוע קבלה: צומת v_i מקבל הודעה, צריך להקדים אותו אירוע שליחת הודעה.

זמן אירוע: אירוע שהופעל בצומת על ידי שעון מקומי כלשהו.

לוחות זמנים וביצועים

תצורות configuration c: ווקטור של כל ה n צמתים בזמן נתון.

קטע ביצוע: רצף של תצורות ואירועים מתחלפים.

ביצוע: קטע ביצוע שמתחיל בתצורה הראשונית c_0.

לוח זמנים: ביצוע ללא תצורות, אך כולל כניסות (רצף אירועי הביצוע והכניסות).

מודל העברת הודעות– הערות

מצב פנימי: המצב הפנימי של צומת לא כוללת את המצב של ההודעה שנשלחה על ידיו.

יריב: במסגרת הנחות הסינכרון, הביצוע/לוחות הזמנים נקבעים בצורע הגרועה ביותר (על ידי היריב).

צמתים דטרמיניסטיים: במודל הבסיסי אני מניחים כי הצמתים הם דטרמיניסטים. בחלק מהמקרים אנו נניח שהצמתים יכולים להטיל מטבע, אלגוריתמים אקראיים. פרטי הדגם/ היריב מסובכים יותר.

Schedule לוח זמנים

לוח זמנים הוא רצף מאורעות שהתרחשו במערכת:

$S = \{s_{12}, s_{23}, r_{21}, s_{31}, r_{32}, s_{13}, r_{13}, r_{31}\}$

איך קוראים את זה ? S_{12} משמעו שקודקוד 1 שולח הודעה לקודקוד 2.

r_{21} נשמשעו שקודקוד 2 קיבל הודעה מקודקוד 1.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Local schedules

$S|v_i$ מה קודקוד i רואה מתוך S. כלומר, כל השליחות והקבלות שהתרחשו בקודקוד i.

לדוגמה: $S = \{s_{12}, s_{23}, r_{21}, s_{31}, r_{32}, s_{13}, r_{13}, r_{31}\}$

$S|v_1 = s_{12}, r_{13}, s_{13}, r_{12}$.

באופן דומה עבור שאר הקודקודים.

Indistinguishability

משפט: יהיו S ו S' שני לוחות זמנים ויהי v_i קודקוד.

נניח ש $S|v_i = S'|v_i$ ול v_i קלט זהה ב S, S'. אז v_i מבצע אותן פעולות ב s, s'.

הוכחה: המצב של קודקוד v_i תלוי רק בקלט וב $S|v_i$. עבור קודקודים דטרמניסטים, הפעולה הבאה תלויה רק במצב הנוכחי.

Asynchronous Executions

יש הגבלות מינימליות על מתי הודעות נשלחות ומתי חישובים פנימיים נעשים.

לוח זמנים נקרא קביל אם:

1. יש אינסוף של מצבי חישוב עבור כל צומת.
2. כל הועדה מועברת בסופו של דבר.

הערות:

1 ו 2 הם תנאי הגינות.

מניחים שצמתים אינם מסתיימים במפורש.

מצב חלופי: לכל צומת יש אינסוף שלבי חישוב או שהוא מגיע למצב עצירה מפורש.

<u>תרגול 1 – סיכום</u>

הקורס מדבר על סנכרון בין מחשבים עם ערוצי תקשורת.

מודלים: שליחת הודעות וזיכרון משותף.

שליחת הודעות

גרף שבו כל קודקוד מייצג מחשב. בכל שלב מתבצעת שליחת הודעה אחת מכולם לכולם.

במערכת סינכרונית: ניתן לדעת את הזמנים. ישנם זמנים מוגדרים לכל דבר.

במערכת אסינכרונית: אין וודאות לגבי הגעה של הודעות, אין הגדרות לזמנים.

זיכרון משותף

המחשבים עובדים לפי אלגוריתם וכותבים/קוראים מהזיכרון.

הם יכולים כמובן גם לעשות עיבוד פנימי.

מושגים

safety: שלא קורה שום דבר רע.

Liveness:קורה משהו טוב, לפחות למישהו אחד.

Fairness: קורה משהו טוב לכולם.

סיכום קורס מערכות מבוזרות

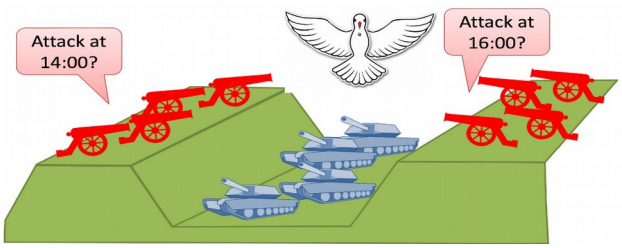
נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

הרצאה 2 – בעיית שני הגנרלים

הדבר שהופך סתם מערכת עם הרבה מחשבים שעובדים באותו הזמן למערכת מבוזרת זה היכולת שלהם לתקשר אחד עם השני. היסוד הכי בסיסי של שיתוף פעולה זה הסכמה.

בעיית שני הגנרלים



בבעיה לצורך המחשה אנו מדברים על שני גנרלים כמובן שניתן להשליך בעיה זו על כל שני תהליכים שרוצים להסכים על משהו.

נניח שיש לנו 2 צבאות, כחול ואדום, הכחול הוא הרבה יותר חזק מהצבא האדום והוא נמצא בעמק, והאדום נמצא על הגבעות ויכול לתקוף משני הכיוונים. הגנרלים של הצבא האדום, החלש יותר צרכים לתאם ביניהם את ההתקפה אם הם יצליחו לתקוף באותו הזמן הם ינצחו, אחרת יפסידו. כל צד חושב שצריך לתקוף משעה מסויימת ועיליהם להגיע להסכמה בנוגע לשעת התקיפה. הדרך שלהם לתקשר היא על ידי שליחת יונים מצד לצד, נניח על ידי קשירת פתק לרגל היונה. התקשורת הזאת לא בטוחה כי הצבא הכחול שלמטה עלול לפגוע ביונים, ואין אנו יודעים האם היונה הגיעה ליעדה או לא (המרחק בין הגבעות גדול). המטרה שלנו להגיע להסכמה בנוגע לשעת התקיפה וכך ללנצח את הצבא הכחול.

זהו משל לכל בעיה יום יומית שצריך להגיע בא להסכמה. למשל שני אנשים שרוצים להפגש ומתקשרים ביניהם בהודעות בתקשורת לא אמינה.

בצורה פורמאלית:

המודל:

- 2 קודקודים המיוצגים על ידי גרף ויש ביניהם קשת דו כיוונית.
- המערכת סנכרונית, ושליחת ההודעות אינה אמינה (יכולות להיות תקלות).

קלט:מה שאנו רוצים שנחליט 0 או 1.

פלט: ההחלטה, 0 או 1

הסכמה: ששני הקודקודים פולטים אותו מספר.

סיום: התוכנית צריכה להסתיים במזן סופי (לא חייב להיות יעיל).

במערכת סנכרונית נמדוד את הזמן לפי מספר הסיבובים.

זוהי למעשה מערכת מבוזרת הכי קטנה שיכולה להיות, ועד כה יש פתרון טרייויאלי לבעיה, ניתן להחליט שתמיד פולטים 0 או 1 לחילופין.

ולכן נוסיף את תנאי הבא:

- validity (אמינות): צריך יותר מהסכמה, אם הקלטים שונים, צריך להסכים על אחד מהם. אבל אם שני הקלטים יהיו 1 או 0, את נצטרך לפלוט 1 או 0 בהתאמה.

שאלות אפשריות: כמה זמן נחכה להודעה במקרה שהיא לא מגיעה? מה עושים במצב שהודעה לא מגיעה? האם ניתן לפתור את הבעיה אם הודעה אבדה?

הבעיה הזאת בלתי פתירה! איך נוכיח זאת? נראה שניתן להכשיל כל אלגוריתם שקיים.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

דמיון הרצות: נקרא לשני הרצות E_i E_j דומות אם הן בלתי ניתנות להבחנה עבור איזשהו קודקוד.
ונסמן $E_i \sim vEj$
הבחנה: בהנחה שהגענו להסכמה, שני הפלטים יהיו 0 או שני הפלטים יהיו 1.
הוכחה: עבור שני הרצות דומות $E_i \sim vE_{i-1}$, קודקוד V יעשה אותה החלטה, ובגלל הסכמה גם u יוציא אותו פלט, וכך עבור כל שתי הרצות סמוכות דומות.

Nodes always decide after exactly T rounds

Execution E_0: Both inputs are 0, no messages are lost.

Execution E_1: One of the messages in round T is lost.

...

Execution E_{2i}: Both messages in round T+1-i are lost.

Execution E_{2i+1}: One of the messages in round T-i is lost.

...

Execution E_{2T}: Both inputs are 0, no messages are delivered. All outputs are 0 due to similarity.

Execution E_{2T}: Both inputs are 0, no messages are delivered. All outputs are 0 due to similarity.

Execution E_{2T+1}: Input of v_1 is 0 but input of v_2 is 1. No messages are delivered.

Execution E_{2T+2}: Both inputs are 1, no messages are delivered.

...

Execution $E_{2T+2i+1}$: Exactly one of the messages in round i is delivered.

Execution $E_{2T+2i+2}$: Both messages in round i are delivered.

...

Execution E_{4T+2}: Both messages in round T are delivered. Decision must be 1 - a contradiction.

סיכום ההוכחה:
מתחילים בהרצה שבה 2 הקודקודים בעלי קלט 0 ושום הודעה לא אבדה. מ validity שני הקודקודים צרכים לפלוט 0. נוריד הודעות אחת אחרי השניה (מהסוף) כדי שיהיה רצף הרצות כך שכל 2 הרצות רצופות הן דומות. מהרצה שבה שום הודעה לא הגיעה, והקלט הוא 0 אפשר להגיע להרצה שבה שום הודעה לא הגיעה והקלט הוא 1. על ידי הוספת ההודעות אחת אחרי השניה מההתחלה לסוף, נגיע להרצה שבה שני הקודקודים בעלי קלט 1 ושום הודעה לא אבדה. מ validity שני הקודקודים צרכים להחליט 1. תחילה שני הקלטים הם 0 לאחר מכן קלט אחד 0 והשני 1 ולבסוף שניהם 1 והפלט הוא 1 – סתירה.

<u>תרגול 2 – סיכום</u>

- תזמון: כל תהליך מבצע בכל שלב את הפעולות הבאות–קבלה, פעולות פנימיות ושליחה.
- קונפיגורציה: מצב נוכחי כל תהליך, מה יש בקלט איפה בזיכרון וכדומה.
- קונפיגורציה של המערכת: ווקטור שבו יש קונפיגורציה של כל תהליך i.
- מהי הרצה? תיאור הפעולות של תהליך, קפיצה בין קונפיגורציה לקונפיגורציה.
- בלתי ניתנים להבחנה: S ו S' יהיו בלתי ניתנים להבחנה אם קיים קודקוד V כלשהו שעבורו זאת אותה הרצה כלומר מקבל אותו קלט באותו הזמן ופולט פלט באותו הזמן.
- הרצה דומה: $E_1 \sim vE_2 \iff E_1|v = E_2|v$

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

הרצאה 3 – בעיית שני הגנרלים

בעיית שני הגנרלים יכולה להיות פתירה אם:

- נסכים לאחד מהגנרלים להטיל מטבע.
- ההסכמה תהיה בהסתברות של אחד מינוס אפסילון.(כאשר ניתן לשלוט על גודל האפסילון).

אלגוריתם הרמות The level algorithm

1. שני הקודקודים מאפסים את הרמה שלהם ל 0.
2. בכל סיבוב: שני הקודקודים שולחים את רמתם אחד לשני.
3. כאשר קודקוד u בעל רמה l_u מקבל הודעה מקודקוד v בעל רמה l_v , קודקוד u מעדכן את הרמה שלו ל:

$$l_u = \max\{l_u, l_v + 1\}$$

הבחנה: הרמה של קודקוד לא יורדת.

טענה: בכל זמן, שני הרמות נבדלות זו מזו באחד לכל היותר.

הוכחה: באינדוקציה על מספר הסיבובים.

בהתחלה $l_u = l_v = 0$

נניח כי הטנה נכונה עבור t סיבובים ונשקול את הסיבוב ה t+1.

אם $l_u = l_v$, הטענה מתקיימת מכיוון שהרמה של כל קודקוד בסיבוב ה t+1 תגדל בלכל היותר 1.

נניח בה"כ ש $l_u = l_v + 1$ לא משנה מה יקרה בסיבוב t+1 קודקוד l_u לא ישתנה. אם קודקוד v יקבל הודעה מ u, אז קודקוד v יעדכן את רמתו: $l_v = l_u + 1$.

טענה: אם כל ההודעות מגיעות, שני הרמות שוות למספר הסיבובים.

הוכחה: באינדוקציה על מספר הסיבובים.

הטענה נכונה בהתחלה.

נניח כי הטנה נכונה עבור t סיבובים($l_u = l_v = t$) ונשקול את הסיבוב ה t+1.

יהיו l_u' l_v' מספרים המציינים את הרמות של קודקודים u,v מייד לאחר סיבוב t+1 לפי תיאור האלגוריתם:

$$l_u' = l_v + 1 = t + 1$$
$$l_v' = l_u + 1 = t + 1$$

טענה: הרמה l_u של קודקוד u שווה ל 0 אם ורק אם u לא קיבל הודעות כלל.

הוכחה: אם u לא קיבל אף הודעה, אז הרמה שלו לא משתנה מתחילת הריצה ולכן היא 0.

אם u קיבל הודעה מקודקוד v כלשהו בעל דרגה l_v מכיוון שהדרגה לעולם לא פוחתת אז $l_v >= 0$

ולכן הרמה של קודקוד u מתעדכנת ל $l_u = \max\{l_u, l_v + 1\}$ שזה מספר גדול ממש מ–1.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

סיכום אלגוריתם הרמות ל 2 קודקודים:

אם האלגוריתם רץ r סיבובים אזי:

1. בסוף הרמות נבדלות זו מזו בלכל היותר 1.
2. אם כל ההודעות הגיעו הרמה של הקודקודים בסיבוב r היא r.
3. הרמה של קודקוד u היא לפחות 1 אם ורק אם u קיבל לפחות הודעה אחת.

בעיית שני הגנרלים – רנדומלי

נניח בה"כ כי קודקוד u יכול להשתמש ברנדומליות.

נניח שהקלטים האפשריים הם $\{0,1\}$.

1. קודקוד u מגריל מספר t מתוך $\{1,.....,r\}$ באופן אקראי אחיד.
2. שני הקודקודים מריצים את אלגוריתם הרמות r פעמים
   א. בזמן ריצת האלגוריתם קודקוד u ישלח את הרמה, קלט, t שלו. וקודקוד v ישלח את הרמה והקלט שלו.
3. בסוף, קודקוד מחליט 1 אם:
   א. הקודקוד יודע מהו t ושני הקלטים נצפו.
   ב. שני הקלטים שווים ל−1.
   ג. רמת הקודקודים היא לפחות t.
4. אחרת החזר 0.

טענה: אם לפחות אחד מהקלטים הוא 0, אז שני הקודקודים יפלטו 0.

הוכחה: מיידית מהאלגוריתם.

טענה: נניח כי שני הקלטים הם 1:

1. אם אף הודעה לא הלכה לאיבוד שני הקודקודים יפלטו 1.
2. שני הקודקודים יחזירו את אותו הערך חוץ מכאשר:

$$\{l_u, l_v\} = \{t-1, t\}$$

הוכחה: נניח ש $\{l_u, l_v\} = \{t-1, t\}$. לפי תאור האלגוריתם, u מחליט 0. מכיוון ש l_v גדול מ 0 ו vמכיר את t ואת שני הקלטים ולכן הוא מוציא 1.

כעת נניח ש $\{l_u, l_v\} != \{t-1, t\}$ אם $l_u, l_v < t$ אז שניהם יחליטו 0.

אחרת $l_u, l_v > t >= 0$ שני הקודקודים מכירים את הקלטים ואת t ולכן יפלטו 1. אם שום הודעה לא הלכה לאיבוד אז $l_v = l_u = r >= t$ ולכן שני הקודקודים יפלטו 1.

משפט: האלגוריתם משיג הסכמה בהסתברות של לפחות:

$$1-1/r$$

הוכחה: ניזכר בכך שהרמות נבדלות אחת מהשניה באחד לכל היותר. מאחר והרמות תלויות אך ורק בהודעות שנשלחו/הגיעו, היריב יכול לוודא שהרמות יהיו $\{l_u, l_v\} = \{i-1, i\}$ עבור i כלשהו מתוך $\{1,....,r\}$. מאחר והיריב אינו יודע מהו t , ההסתברות ש $\{l_u, l_v\} = \{t-1, t\}$ היא אחד חלקי r. משני הטענות האחרונות הקודקודים יגיעו להסכמה בכל מקרה אחר. ולכן הסיכוי לכישלון הוא אחד חלקי r.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

חסם תחתון על הסיכוי לכישלון

נשתמש בטכניקה דומה לבעיית שני הגנרלים בבעיה הדטרמיניסטית ניתן להודיע גבול תחתון על הטעות. נוכיח עבור גרסה חזקה יותר של הבעיה:

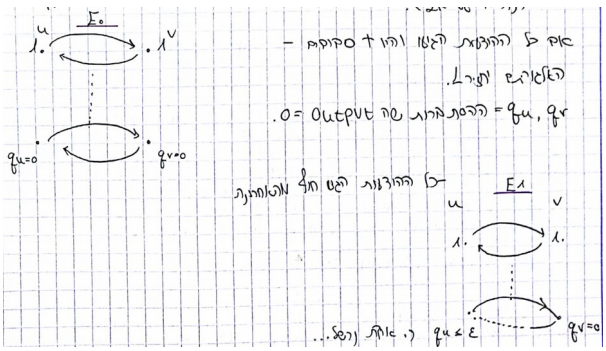
אם לפחות אחד הקלטים הוא 0, שני הקודקודים יפלטו 0.

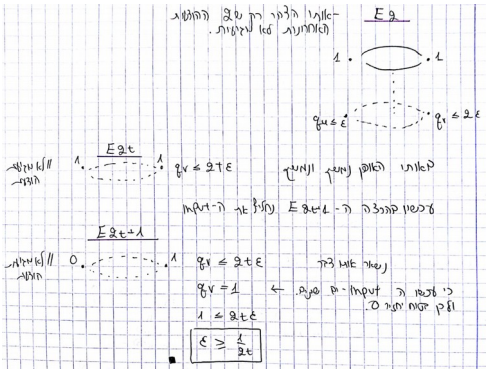
כדי להוכיח את החסם התחתון, נניח שאם שני הקלטים 1 אז:

- אם אף הודעה לא הלכה לאיבוד, אז הפלט חיים להיות 1.
- אחרת, הקודקודים צרכים להוציא פלט זהה בהסתברות של אחד פחות אפסילון.

משפט: מגרסה החזקה של בעיית שני הגנרלים, אם הצמתים צרכים להחליט בתוך r סיבובים, אז ההסתברו אפסילון לאי הסכמה על אותו ערך הוא קטן שווה לאחד חלקי r.

הוכחה:





<u>הרצאה 4 – אלגוריתמי שידור ברשתות</u>

העברת הודעות בטופולוגיות שרירותיות

טופולוגיית הרשת ניתנת על ידי גרף לא מכוון $G=(V,E)$.

הודעות אסינכרוניות עוברות ללא כשלים:

1. ההודעות תמיד עוברות בזמן חסום.
2. עיכובים בדרך כלל אינם ניתנים לחיזוי.

האלגוריתמים מבוססים על אירועים:

1. עם קבלת הודעה משכן.
2. בצע כמה פעולות חישוביות מקומיות.
3. שלח הודעה לשכנים.

Broadcast

צומת מקור v צריך לשדר הודעה M לכל הצמתים במערכת. לכל צומת יש ID יחודי. תחילה, כל צומת מכיר את ה ID של שכינוי.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Flooding

זהו אחד מאלגוריתמי רשת מבוזרת הפשוטים ביותר.

רעיון בסיסי: בעת קבלת הודעה M בפעם הראשונה, העבירו אותה לכל השכנים.

אלגוריתם:

- צומת מקור v: בתחילה תעשה: שלח הודעה M לכל השכנים.
- שאר הצמתים u: עם קבלת הודעה M מקודקוד שכן w: אם M לא נשלח לפני כן, אז תשלח את ההודעה M לכל השכנים פרט ל w.

Flooding במערכת סינכרונית

הזמן מחולק לסיבובים סינכרוניים: עיכוב ההודעה = סיבוב 1.

סיבוכיות: מספר הסיבובים.

התקדמות באלגוריתם flooding:

- אחרי סיבוב אחד: הקודקודים שמכירים את ההודעה M הם בדיוק הקודקודים הרחוקים 1 לכל היותר מקודקוד המקור v.
- לאחר 2 סיבובים: הצמתים שמכירים את ההודעה M הם בדיוק הצמתים הרחוקים 2 לכל היותר מקודקוד המקור v.
- ובאופן דומה עבור שאר הסיבובים.

הרדיוס של גרף $G=(V,E)$: יהיה צומת $v\in V$, הרדיוס של v ב G הוא המרחק המקסימלי מכל שאר הקודקודים בגרף:

$$rad(G,v)=\max\{dist_G(u,v) \mid u\in V\}$$

הרדיוס של G הוא: חשב את הרדיוס עבור כל קודקודי הגרף – המינימלי מביניהם הוא הרדיוס

$$rad(G) = \min\{rad(G,v) \mid v\in V\}$$

הקוטר של G: המרחק המקסימלי בין שני קודקודים בגרף G

$$diam(G)=\max\{dist_G(u,v) \mid u,v\in V\}$$

סיבוכיות זמן של אלגוריתם flooding במערכת סינכרונית הוא $rad(G,v)$.

אבחנה:

$$diam(G)/2 <= rad(G) <= rad(G,v) <= diam(G)$$

סיבוכיות זמן במערכת אסינכרונית

הנחות: עיקובי ההודעות והזמן לחישובים מקומיים הם שרירותיים– האלגוריתמים לא יכולים להניח הנחות תזמון!

לניתוח: עיכובים בהודעות הם לכל היותר יחידת זמן אחת. חישובים מקומיים לוקחים 0 זמן.

קביעת סיבוכיות זמן: קבע את זמן הריצה של כל הרצה נתונה. סיבוכיות זמן ריצה של מערכת אסינכרונית הוא הזמן המקסימלי עבור כל ההרצות.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

זמן הריצה של הרצה

הקצה זמנים לשליחה וקבלת אירועים כך:
סדר האירועים נותר ללא שינוי, חישובים מקומיים לוקחים 0 זמן, עיכובים בהודעות הם לכל היותר 1 יחידת זמן, אירוע השליחה הראשון הוא בשעה 0, זמן האירוע האחרון הוא מקסימלי.
בעקרון: ננרמל את עיכובי ההודעות כך שהעיכוב המירבי הוא יחידת זמן אחת.
הגדרה: סיבוכיות זמן במערכת אסינכרונית, זמן כולל של ההרצה במקרה הגרוע ביותר כאשר חישובים מקומיים לוקחים 0 זמן וכל עיכובי ההודעות הם יחידת זמן אחת לכל היותר.

משפט: סיבוכיות זמן הריצה של אלגוריתם flooding מקודקוד מקור v ברשת אסינכרונית היא $rad(G,v)$.

הוכחה: יהי u€V כך ש $rad(G,v)= dist_G(u,v)$. לפי העיכוב כל ההודעות יגיעו בדיוק ביחידת זמן אחת, היריב יוודא את זה שהזמן שלוקח להודעה M ש v שלח ל u תגיע בלפחות $rad(G,v)$.
לעומת זאת, יהי u€V צומת כלשהו. ויהי $t= dist_G(u,v)$ נשים לב ש $t>= 0$, ו $t<= rad(G,v)$.
נוכיח באינדוקציה על t שהזמן שלוקח ל M להגיע ל u הוא לכל היותר t יחידות זמן.
בסיס: $t= 0$ אזי $u= v$, ולכן v קודקוד המקור מכיר את M.
הנחה: נניח נכונות עבור $t >= 0$, ו $t<rad(G,v)$ ונוכיח עבור $t+ 1$.
עצד: יהי $v, w_1, w_2, ..., w_t, u$ מסלול באורך $t +1$ מ v ל u. לפי הנחת האינדוקציה, ההרצה תגיע לקודקוד w_t בלכל היותר t יחידות זמן. ההודעה M תישלח מ w_t ל u, היא תגיע לקודקוד u לאחר יחידת זמן אחת לכל היותר. לפיכך, הודעה M הגיע לקודקוד u בזמן שלכל היותר $t+ 1$.

סיבוכיות שליחת הודעות באלגוריתם flooding

סיבוכיות שליחת הודעות: המספר הכולל של ההודעות שנשלחו.
משפט: סיבוכיות שליחת ההודעות של flooding בגרף $G= (V,E)$ היא $O(|E|)$.
הוכחה: יהי uw€E צלע כלשהי בגרף G. לפי הגדרת האלגוריתם, u שולח הודעה אחת לכל היותר ל w ו w שולח הודעה אחת לכל היותר ל u. לפיכך, מספר ההודעות המקסימלי הוא $2*|E|$

Flooding spanning tree

אלגוריתם flooding יכול לשמש לחישוב עץ פורש של רשת.
הרעיון:
קודקוד המקור v הוא השורש של העץ.
עבור כל שאר הקודקודים, שכן שממנו התקבלה הודעה M בפעם הראשונה, יהיה האבא של אותו קודקוד.

אלגוריתם:

Source node v:

initially do

parent = NIL (i.e. v is the root of the tree).

send M to all neighbours.

Non-source node u:

upon receiving M from some neighbor w: if M has not been received before, then

parent = w.

send M to all neighbors except w.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

עצים פורשים במערכת סינכרונית

במערכות סינכרוניות, קודקוד u מקבל את ההודעה בסיבוב t אם ורק אם $dist_G(u,v)=t$. לפיכך, בעץ שנבנה המרחק של קודקוד u מהשורש v שווה למספר הסיבובים שלקח ל M להגיע ל u. מכאן שהעץ שנבנה שומר על המרחקים לשורש הגרף G. עצים כאלה נקראים shortest path trees או breadth first search trees.

משפט: במערכות סינכרוניות אלגוריתם flooding בונה עץ BFS.

עץ פורש במערכת אסינכרונית

הבחנה: במערכת אסינכרונית ניתן לבנות כל עץ פורש באמצעות אלגוריתם flooding. בפרט עומק העץ יכול להיות בגודל n−1 גם אם רדיוס/קוטר הגרף הוא 1.

Convergecast

זה ההפך מ broadcast: בהינתן עץ פורש בעל שורש, כל הצמתים שולחים הודעות לשורש.
דוגמה: ניתן לחשב כך את סכום הערכים בעץ מושרש.
אלגוריתם:

Leaf node u:

Initially do: send a message to parent (e.g., send input value).

Internal node w:

Upon receiving a message from child node x: **if** w has received messages from all children, **then** send message to parent (e.g., send sum of all inputs in the subtree whose root is w).

Root node v:

Upon receiving a message from child node x: **if** v has received messages from all children, **then** convergecast terminates.

סיבוכיות זמן: כעומק העץ.
סיבוכיות הודעות: מספר הקשתות בעץ.

Flooading Echo Algorithm

בהינתן שורש, ניתן להשתמש יחד באלגוריתם flooding ו convergecast באופן הבא:

1. השתמש ב flooding כדי לבנות את העץ.
2. השתמש ב convergecast כדי לדווח לשורש בחזרה על סיום.

סיבויות זמן של flooding +convergecast: עומק העץ.
במערכת סינכרונית $O(diam(G))$
במערכת אסינכרונית $O(|V|)$
הערה: במערכת אסינכרונית עומק העץ שנבנה יכול להיות $\Omega(|V|)$, אפילו אם קוטר העץ הוא 1.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

הרצאה 5 – אלגוריתמי שידור ברשתות

Constrictong shortest path tree

Dijkstra algorithm:

Grow tree from source node v.

At intermediate step t, subtree of all nodes at distance at most r_t from source node v.

Next step: add to the tree some node whose distance to v is minimal.

Bellman-Ford algorithm:

Every node u maintains a distance estimate d_u to the source node v.

Initially: $d_v = 0$ and $d_u = \infty$ for every $u \neq v$.

In each step, all nodes update their estimate based on neighbour estimates:

$d_v = \min \{d_v, \min \{d_u + 1 : u \in N(v)\}\}$.

דייקסטרה מבוזר

במקרה שלנו הגרף אינו ממושקל. לכן אנחנו יכולים לבנות את העץ רמה אחרי רמה, כמו במערכת סינכרונית. נניח שהעץ נבנה עד למרחק r מהשורש, קודקוד המקור v.

איך נוכל להוסיף את הרמה הבאה?

קודקוד המקור יבצע את הפעולות הבאות (שלב r+ 1):

1. קודקוד המקור שולח broadcast "התחל את שלב r+ 1" בעץ הנוכחי.
2. העלים בעץ הנוכחי (ברמה r) שולחים "הצטרפו כרמה r+ 1" לשכנים החדשים.
3. קודקוד u מקבל הודעה "הצטרפו כרמה r+ 1" משכנו w:
   1. הודעה ראשונה כזו: w יהיה האבא של u ו–u שולח ACK ל w.
   2. אחרת, u שולח NACK ל w.
4. אחרי קבלת ACK או NACK מכל השכנים, הקודקודים שברמה r מודיעים לשורש על ידי הפעלת convergecast.

סיבוכיות זמן: $O(1+ 2+ \ldots + rad(G,v)) = O(diam(G)^2)$

סיבוכיות הודעות:

$$O(|E| + |V| * diam(G))$$

Bellman–Ford **מבוזר**

רעיון מרכזי:

כל קודקוד u שומר מספר טבעי d_u שמייצג את המרחק המשוער שלו מהשורש v.

בכל עת קודקוד u יכול לשפר את d_u , ולאחר מכן להודיע על זאת לכל שכניו.

אלגוריתם:

1. **Initialization:** $d_v = 0$. For any node $u \neq v$, $d_u = \infty$ and parent(u) = NIL.
2. Root v sends "1" to all its neighbours.
3. For all other nodes u: upon receiving message "t" with $t < d_u$ from neighbour w
   a. set $d_u = t$.
   b. set parent(u) = w.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

משפט: סיבכיות זמן של בלמן פורד מבוזר הוא $rad(G,v) = O(diam(G))$.
הוכחה: יהיה u שייך ל V קודקוד כלשהו, כך ש $rad(G,v) = dist(u,v)$. על ידי עיכוב כל ההודעות, כל ההודעות יעוכבו ביחידת זמן אחת, היריב יוודא שהפעם הראשונה ש u יקבל הדועה כלשהי היא לפחות $rad(G,v)$. ומנגד, יהי u שייך ל V קודקוד כלשהו ויהי $t= dist_G(u,v)$. נוכיח באינדוקציה על t ש $t= d_u$.
בסיס: $t= 0$ קל לראות לפי האתחול של האלגוריתם ש $d_v= 0$.
הנחה: נניח נכונות עבור t בין 0 ל $rad(G,v)$ ונוכיח עבור $t+1$.
יהי $vw_1.....w_tu$ מסלול באורך $t+1$ מ v ל u. לפי הנחת האינדוקציה בכל הרצה $d_{wt} = t$ ביחידת זמן t.
אחרי יחידת זמן אחת לכל היותר u יקבל הודעה "$t+1$" מ w_t ואז u מעדכן את המרחק ל $d_u= t+1$.

משפט: סיבוכיות שליחת הודעות בבלמן פורד מבוזר היא: $O(|E|*|V|)$
הוכחה: די להוכיח שעבור כל זוג קודקודים u,w שייכים ל V כאשר uw שייך ל E, ש u שולח לכל היותר n הודעות ל w. כל הודעה שנשלחת מ u ל w היא המרחק המשוער של u מהשורש, הודעה זו מכילה מספר טבעי בין 0 ל $n-1$. אם u שולח ל w הודעה "t1" ולאחר מכן שולח הודעה "t2" אז $t2<t1$. ולכן לכל היותר ישלח n הודעות.

הרצאה 6 – Causality, Logical Time and Global States

שעונים לוגיים Logical clocks
המטרה: הקצאת חותמת זמן לכל האירועים במערכת העברת הודעות אסינכרונית.
דבר זה יאפשר לנו לתת לצמתים מושג כלשהו של זמן.
ערכים של השעון הלוגי: ערכים מספריים שעולים בהתאמה להתנהגות הצפוייה של המערכת.
המטרה כאן היא לא לבצע סנכרון של שעון.
סנכרון שעון: מחשבים שעונים לוגיים בכל הצמתים המדמים זמן אמיתי ואשר מסונכרנים היטב.

התנהגות הניתנת לצפייה Observal behavior
תזכורת:
הרצה היא רצף לסירוגין של configurations and event.
לוח זמנים S הוא רצף האירועים של הרצה.
לוח הזמנים של צומת v, מצויין על ידי $S|v$, הוא רצף האירועים ב S שנצפו על ידי קודקוד v.

Causal shuffles: לוח זמנים S' הוא Causal shuffle של לוח זמנים S אם $s|v = s'|v$ עבור v שייך ל V.
הבחנה: מערכת מבוזרת לא יכולה להבדיל בין לוחות זמנים S ו S' אם ורק אם S הוא Causal shuffle של S'.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Causal order

הגדרה: בלוח זמנים S, אירוע e התרחש ככל הנראה לפני אירוע e' אם ורק אם e מופיע לפני e' בכל Causal shuffle של S.

שעונים לוגיים מבוססים על סדר סיבתי של אירועים.

אם אירוע e מתרחש לפני אירוע e', אזי e אמור להופיע לפני e' ב causal order.

במקרה כזה השעון של e אמור להיות קטן מהשעון של e'.

Lampot happens before relation

הנחה: מערכת שליחת הודעות, וישנם אירועים של שליחה וקבלת הודעות.

נניח ששני אירועים e ו-e' בלוח זמנים S מתרחשים בקודקודים u ו u' בהתאמה.

- אירוע שליחה מתרחש כאשר הקודקוד שולח הודעה ואירוע קבלה מתרחש כאשר קודקוד מקבל הודעה.
- e קורה בזמן t ו e' קורה בזמן t'.

בכל אחד מן המקרים הבאים אנו יודעים ש e provably occurs before e' :

1. t'>t and u'= u (אותו קודקוד והזמן של e קטן יותר).
2. e' הוא אירוע קבלה של שליחת אירוע e.
3. קיים אירוע e'' שאנחנו יודעים ש e provably occurs before e'' in S ו-e'' provably occurs before e' in S.

הגדרה: ה $=>_s$ happens before relation מעל לוח זמנים S הוא יחס בינארי על שליחת/קבלת אירועים ב S וזה מכיל:

1. כל הזוגות (e,e') כך ש e מקדים את e' ב S ושני האירועים התרחשו באותו הקודקוד.
2. כל הזוגות (e,e') כך ש e אירוע שליחת הודעה ו e' אירוע קבלה של אותה הודעה.
3. כל הזוגות (e,e') עבורם יש אירוע שלישי e'' כך ש

$$e=>_s e'' \ , \ e''=>_s e'$$

Happens before and causal shuffle

משפט: עבור לוח זמנים S ו 2 אירועים (שליחה/וקבלה) e,e' שני המשפטים הבאים שקולים:

1. אירוע e happens before e' in S. כלומר: $e=>_s e'$
2. אירוע e מקדים את אירוע e' בכל S' שהוא causal shuffle של S.

הערה: המשפט הנ"ל מראה ש happens before תופס בדיוק את הסיבתיות בין האירועים, כלומר הוא תופס את כל מה שקשור לסדר האירועים הניתנים לראות מתוך המערכת.

Lamport clocks

רעיון ובטיחות:

1. כל אירוע e מקבל ערך שעון T(e) שייך ל N (טבעיים).
2. אם יש e ו e' אירועים באותו קודקוד ו e מקדים את e' אזי: T(e)<T(e').
3. אם s_m , r_m הם אירועים של שליחה וקבלת אותה הודעה M, אזי: $T(s_M)<T(r_M)$.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

הבחנה: כפי שהיחס "קטן" מעל הטבעיים הוא טרנזיטיבי, עבור ערכי השעון המקיימים את 1,2,3 נקבל ש :

$$e=>_s e' \quad \rightarrow \quad T(e)<T(e')$$

מכאן שהיחס החלוקה שהוגדר על ידי T הוא $=>_S$ superset of.

אלגוריתם:

- Each node u maintains a counter c_u which is initialized to 0.
- For any event e at node u which is not a receive event, node u sets $c_u = c_u + 1$ and then sets $\tau(e) = c_u$.
- For any send event s occurring at node u, node u includes the value of $\tau(s)$ in the sent message.
- For any receive event r at node u (with the corresponding send event s), node u sets $c_u = \max\{c_u, \tau(s)\} + 1$ and $\tau(r) = c_u$.

הסבר:

כל קודקוד מחזיק מונה.

1. בשליחה תעלה את ערך המונה ב 1.
2. בקבלה עדכן את ערך המונה להיות: max{מונה, מונה שבהודעה} +1

Fidge mattern vector clocks

שעון למפורט מעניק סופרסט של היחס happens before.

האם אנחנו יכולים לחשב שעונים לוגיים שיניבו את happens before במדוייק?

Vector clocks:

כל קודקוד u שומר וקטור של ערכי שעון, ערך אחד עבור על קודקוד v שייך ל V.

בוקטור של הקודקוד u אשר מוקצה לאירוע כלשהו e שמתרחש בקודקוד u, הכניסה $Vc_v(u)$ מתאימה לקודקוד v המייצג את מספר האירועים שהתרחשו בצומת v ש u מודע להם בזמן אירוע e, כלומר, ווקטור קלוק של קודדוד u בקואורדינטה v.

אלגוריתם:

- Each node u maintains a vector $VC(u)$ whose entries correspond to the nodes in V, where
  - all entries are initialized to 0.
  - The entry corresponding to node $v \in V$ is denoted by $VC_v(u)$.
- For any event e at node u which is not a receive event, node u sets $VC_u(u) = VC_u(u) + 1$ and then $VC(e) = VC(u)$.
- For any send event s occurring at node u, node u includes the value of $VC(s)$ in the sent message.
- For any receive event r at node u (with the corresponding send event s), node u sets
  - $VC_v(u) = \max\{VC_v(u), VC_v(s)\}$ for every $v \neq u$.
  - $VC_u(u) = VC_u(u) + 1$.
  - $VC(r) = VC(u)$.

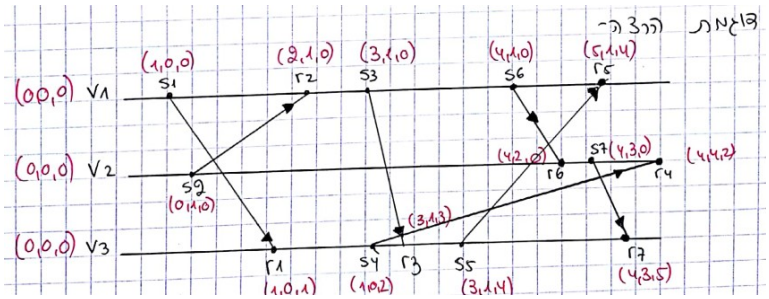
סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

הסבר:

- כל קודקוד מחזיק ווקטור שעונים, קואורדינטה עבור כל קודקוד במערכת.
- כל המספרים בוקטור מאותחלים ל 0.
- במאורע שליחה: נעלה את ערך הקואורדינטה שלי באחד, ונשלח לקודקוד השני את כל הווקטור שלי.
- באירוע קבלת הודעה: הווקטור של שולח ההודעה נמצא בהודעה. דבר ראשון, בוקטור שלי בקואורדינטה נעלה את הערך באחד. עבור כל קודקוד אחר בווקטור, לוקחים את המקסימום בין הערך בכל קואורדינטה בוקטורים ומעדכנים את הווקטור שלי למקסימום בכל תא.



Vector clocks and happens before

Definition: For two events e and e' we say that $VC(e) < VC(e')$ if $VC_v(e) \le VC_v(e')$ for every $v \in V$ and $VC(e) \ne VC(e')$.

Theorem: Given a schedule S, for any two events e and e' we have $e \Rightarrow_S e' \leftrightarrow VC(e) < VC(e')$.

הרצאה 7 – Causality, Logical Time and Global States

Logical clock vs. Synchronizers

פעימות השעון הנוצרות על ידי הסנכרוניזר נראים גם כשעונים לוגיים.

- שליחת אירועים בסיבוב t מוקצים לערך שעון של 2t−1.
- קבלת אירועים בסיבוב t מוקצים לערך שעון 2t.

תכונות:

- סופרסט של היחס hapens before.
- דורש שינוי דרסטי בפרוטוקול והתנהגותו: המסנכרן קובע מתי ניתן לשלוח הודעה.
- שיטה בעלת משקל כבד מאוד להשגת ערכי שעון לוגיים: דורש הרבה הודעות.

ישום של זמנים לוגיים

replicated state machine: יש צורך לסנכרן בין מכונות דומות. לדוגמה, נניח ויש אתר איטרנט שמוכר שעונים, והאתר מחליט על מבצע, הנחה ל 100 הראשונים. איך נדע מי הם 100 הראשונים?

פתרון:

הוספת ערך השעון הנוכחי לכל פעולה. יש לבצע את הפעולות לפי ערכי השעון.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Global state

לפעמים הצמתים של מערכת מבוזרת צרכים להבין את מצבה הגלובלי של המערכת. למשל, על מנת לדעת מה ערכו של מאפיין כלשהו של המערכת.

- לא ניתן לבחין במערכת בין הרצות/לוחות זמנים בעלי יחס hapens before זהה.
- באופן כללי, לא ניתן לרשום את ה global state בכל זמן נתון של ההרצה.

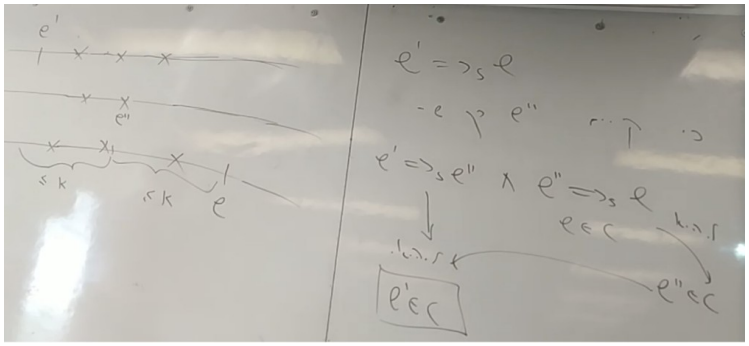
הפתרון הטוב ביותר האפשרי: ציון ה global state התואמת את כל התצוגות המקומיות. כלומר, מצב שניתן להשגה באותו זמן. (תיאור רגעי בכל קודקוד של כל האירועים שהתחרשו עד לאותו הרגע)
פתרון זה נקרא consistent or global snapshot של המערכת והוא מבוסס על consistent cuts של לוח הזמנים.

Consistent cuts

הגדרה: בהינתן לוח זמנים S, נגדיר **cut** הוא subset C של האירועים ב S, כך שלכל קודקוד v שייך ל V, האירועים ב C המתרחשים ב v, הם תחילית של רצף האירועים ב S|v.
כלומר, חתך ב-S הוא prefix של כל הקודקודים.
הגדרה: בהינתן לוח זמנים S, נגדיר C **Consistent cut** כ cut כך שעבור כל אירוע e€C ולכל אירוע n e'€S, מתקיים ש:

$$e' =>_S e \;\; -> \;\; e' €C$$

טענה: בהינתן לוח זמנים S, חיתוך C הוא consistent cut אם ורק אם אם עבור כל הודעה M עם אירוע שליחת הודעה S_M ואירוע קבלה r_M, אם r_M€C אזי s_M€C.
הוכחה: יהי C חיתוך שהוא consistent, תהי M הודעה בעלת אירוע שליחה S_M ואירוע קבלה r_M ב C.
לפי ההגדרה של היחס happens before, אנו יודעים ש $S_M =>_S r_M$ מכאן ש S_M€C מהגדרת חתך עקבי.
כיוון שני: עלינו להראות כי C הוא עקבי. למעשה e'€C מתקיים בכל פעם ש e'€S, e€C ו $e' =>_S e$.
אם e ו e' הם אירועים באותו קודקוד, זה נכון עבור כל חיתוך.
אם e ו e' הם אירוע שליחה וקבלת אותה הודעה, מתקיים מההנחה.
אחרת ניתן להראות באינדוקציה על מספר האירועים בין e' ל e.



Consistent snapshot

consistent snapshot = global snapshot = consistent global state
consistent snapshot: הוא מצב גלובלי של המערכת שהוא עקבי עבור כל מבט מקומי.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Global system state: וקטור של מצבים מידיים של כל הקודקודים ותיאור של כל ההודעות שהועברו עד כה.

טענה: ווקטור עם קואורדינטה לכל קודקוד הוא consistent snapshot אם ורק אם הוא תואם את מצבי הצומת של איזה חתך קבוע.

Proof: Let V be a consistent snapshot. Note that V must correspond to a cut **C** as otherwise some node would be able to tell the difference. Suppose for a contradiction that **C** is not consistent. Then, by a previous claim, $r_M \in$ **C** and $s_M \notin$ **C** for some message M. Clearly, this does not correspond to any system state.

Conversely, let **C** be a consistent cut and, for every node u, let t_u be some time after the last event in **C**$\cap s|u$ and before the first event in $s|u \setminus$ **C**. Let t = max{t_u : $u \in$ V} and let t' = min{t_u : $u \in$ V}. For every $u \in$ V, we shift everything that happened at time t_u or later, forwards by time t - t' and cut u at time t. Clearly, we don't change the order of events at any node. Moreover, since **C** is a consistent cut, for every message M, if we shift s_M, then we shift r_M by the same amount.

Computing a consistent snapshot

שימוש בשעונים לוגים:

נניח שלכל אירוע e יש ערך שעון T(e) כך שעבור כל שני אירועים e,e' מתקיים:

$$e =>_s e' \quad -> T(e)<T(e')$$

בהינתן T נגדיר $C(T_0)$ כקבוצת האירועים e המקיימת $T(e) <= T_0$. כלומר, ערך סף כלשהו.

טענה: $C(T_0)$ הוא חתך עקבי עבור כל $T_0 >= 0$.

הוכחה: נובע מיידית מההגדרה.

הערות:

זה לא ברור תמיד איך נבחר את T_0.

כאשר T_0 הוא גדול, לא ברור כמה זמן יקח לנו לקבל את תמונת המצב.

כאשר T_0 הוא קטן, תמונת המצב לא תהיה הכי מעודכנת.

Chandy Lamport snapshot algorithm

המטלה: לחשב סנאפשוט עקבי במערכת פועלת.

הנחות:

- אין צורך בשעונים לוגיים.
- הערוצים מקיימים את הנחת ה FIFO.
- אין כשלים.
- הרשת קשירה (רכיב קשירות אחד).
- כל צומת ברשת יכול לבצע snapshot.

הערה: הנחת ה FIFO יכולה תמיד להתקיים, השולח שולח מספור הודעות מקומי עבור כל ערוץ יוצא. יש לעבד הודעות עם מספרים קטנים יותר לפני מספרים גדולים יותר. עובד כל עוד ההודעות לא אבדו.

סקירה כללית:

- נניח כי קודקוד u כלשהו מתחיל בתהליך סנאפשוט.
- מועדי ביצוע החיתוך בצמתים השונים נקבעים על ידי שליחת הודעות מסומנות.
- כאשר אנו מקבלים הודעה מסונת ראשונה, הקודקוד "מקליט" (מבצע חיתוך) את המצב שלו, ושולח הודעות מסומנות לכל השכנים שלו.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

- בכל ערוץ נכנס, ההודעות שהגיעו בין תחילת מצב "ההקלטה" וקבלת הודעה מסומנת (בערוץ זה) נקראות הודעות במעבר.
- לאחר קבלת הודעה מסומנת מכל הערוצים הנכנסים, הקודקוד סיים את תפקידו בסנאפשוט.

האלגוריתם:

Initially: Node u records its state.

When node w receives a marker message from node v: if w has not yet recorded its state, **then**

- w records its state.
- The set of messages in transit from v to w is empty.
- w starts recording messages on all other incoming channels.
- **Else:** the set of messages in transit from v to w is the set of recorded messages since w started recording messages on this channel until now.

(Immediately) after node w records its state: w sends marker messages on all outgoing channels (before sending any other message on those channels).

משפט: האלגוריתם הנ"ל מחשב חתך עקבי והוא מחשב נכון את ההודעות במעבר.

הוכחה: באופן אינטואיטיבי, האלגוריתם מחשב חתך ע. נניח בשלילה כי ע הינו לא עקבי. לפיכך, קיימת הודעה M שקלטה לא יכי קודקוד בשרת v לקודקוד בשרת w, כך ש $m \in C_{vw}$, $m \notin S_{wv}$ לאחר ש $m \notin S_{wv}$, מתקיים ש-v כיבה כבר את התיעוד. לפי אלגו' v שלח הודעה מסומנת ל w לפני שליחת M. לפי C_{vw} בסביבה מתקיים ה FIFO.

לכן ניתן שכל ההודעות שקודקוד בשרת v לקודקוד בשרת w, שסומן w בהודעות מגיעה. לפי אלגוריתם, אלו הם כל ההודעות שקבל w לאחר כיבוי התיעוד, ולכן שהגיע קודם הודעה מסומנת מ-v. לפי הנחת ה FIFO אלו הם בדיוק ההודעות M מ v לשם כך ש $m \in S_{wv}$ לפי $m \notin C_{vw}$ מבטן ההודעות במעבר.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

הרצאות Consensus 8–13

הגדרה: ישנם n תהליכים/חוטים/צמתים $v_1, \ldots, vn$.
לכל צומת v_i יש קלט x_i שנלקח מקבוצה משותפת D.
כל צומת (שלא קרס) v_i מחשב פלט y_i מתוך D.
דרישות:

- הסכמה: הפלט של כל התהליכים שלא קרסו זהה.
- Validity: אם כל הקלטים הם X, אז כל הפלטים יהיו X.
- סיום: כל התהליכים שלא קרסו מסתיימים לאחר מספר סופי של צעדים.

נחזור לבעיית שני הגנרלים, זוהי בעיית הסכמה בינארית בין שני קודקודים.

- מודל: התקשורת היא סנכרונית, ההודעות יכולות ללכת לאיבוד.
- תוקף: אם לא הולכות לאיבוד הודעות ושני הצמתים בלי קלט זהה, על הפלט להיות כמו הקלט של שני הצמתים.

ראינו בעבר שלא ניתן לפתור את הבעיה הזאת עם ההתניות הללו.

Consensus is important

ישנם בעיות טבעיות רבות הדורשות הסכמה. למשל: בחירת מנהיג, פתרון של בעיית שני הגנרלים, הסכמה לגבי הצעד הבא של חישוב משותף.
אנחנו נראה שבחלק מהמודלים, הסכמה היא אפשרית ואילו בדגמים אחרים היא לא. מטרתנו היא להחליט אם הסכמה היא אפשרית למודל נתון ולהוכיח זאת.

Consensus 1: Shared Memory

ישנם n>=2 מעבדים.
זיכרון משותף הוא זיכרון שניתן לגשת אליו בו זמנית על ידי מספר חוטים/תהליכים.
מעבדים יכולים לקרוא או לכתוב לתא זיכרון משותף באופן אטומי אך לא את שניהם.
פרוטוקול: יש תא ייעודי C.
תחילה C במצב מיוחד "?" מעבד 1 כותב את ערכו x_1 אל c ואז מחליט על x_1. מעבד $j \neq 1$ קורא מ c עד שהוא רואה ערך כלשהו שאינו "?" ואז מחליט על ערך זה.
בעיות בגישה זו?

Computability

הגדרת החישוב: חישוב בדרך כלל מתייחס למכונת טיורינג. כלומר, ניתן לפתור את הבעיה הנתונה באמצעות מכונת טיורינג. מודל מתמטי חזק.
חישוב זיכרון משותף: מודל של חישוב אסינכרוני, ניתן לחישוב ללא המתנה במעבד בעל מספר תהליכים(wait–fee).

Consensus 2: Wait-free Shared Memory

ישנם n>=2 מעבדים.
מעבדים יכולים לקרוא או לכתוב לתא זיכרון משותף באופן אטומי אך לא את שניהם.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

מעבדים עלולים להתרסק.

Wait fee implemantation:

כל תהליך מסתיים במספר סופי של שלבים. לא ניתן להשתמש במנעולים כי יכול להיווצר מצב שתהליך מחזיק במנעול וקרס. אנו מניחים שיש לנו אוגרים אטומיים wait free (קריאה/כתיבה לא חופפים).

אלגוריתם:

A designated memory cell c is initialized to "?"

Every processor i runs the following algorithm

r = read(c);

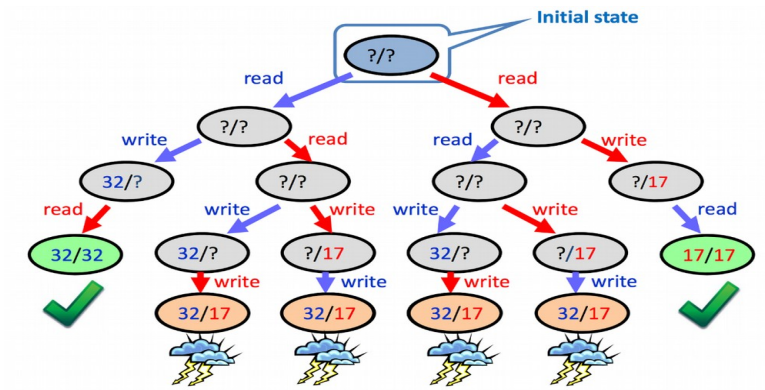
If r = ? **then** write(c, x_i) and decide x_i;

Else decide r.

האם האלגוריתם נכון?

לא, נשים לב למקרה שבו אחד התהליכים נגש לקרוא מה יש באוגר ורואה "?" לכן הוא מחליט להחליף את ה"?" בערך שלו, אך לפני שהוא מספיק לעשות זאת תהליך אחר מגיע לאוגר ורואה גם את ה"?" שטרם הוחלף. ולכן הוא מחליט להחליף גם את ערך האוגר מה- "?" לערך שלו. מה שבפועל יקרה זה שהם ידרסו אחד את הערך של השני. דבר זה מתאפשר מכיוון שקריאה וכתיבה זה לא פעולה אטומית, אלא רק כל אחד מהם בנפרד.

עץ הרצות:

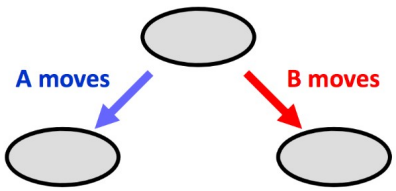


מפשט: אין אלגוריתם קונצנזוס ללא המתנה אסינכרוני המשתמש בקריאה/כתיבה באופן אטומי.

הוכחה: לשם הפשטות, ישנם רק שני תהליכים A ו B והקלט הוא בינארי.

נניח בשלילה שיש פרוטוקול המשיג הסכמה תחת התנאים הנ"ל.

בפרוטוקול זה A ו B "נעים" בכל שלב. מעבר יכול להיות או בעת קריאה מרגיסטר או בעת כתיבה לרגיסטר.

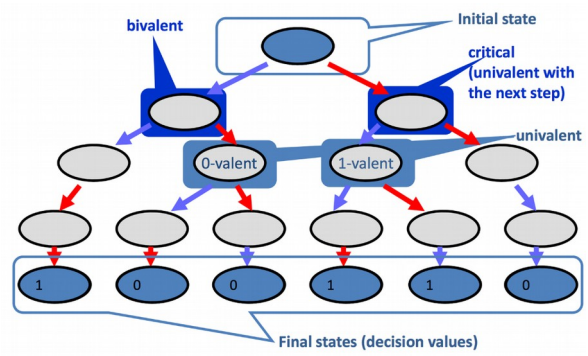


סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Bivalent vs. Univalent



Wait-free computation הוא עץ.

Bivalent זהו מצב שניתן להגיע ממנו גם ל 0 וגם ל 1. כלומר, כאשר אתה נמצא בעץ בקודקוד שהוא Bivalent אזי נוכל להגיע לפלט 0 או 1 בסופו של דבר.

Univalent קודקוד שממנו ניתן להגיע ל 0 או ל 1 בלבד.

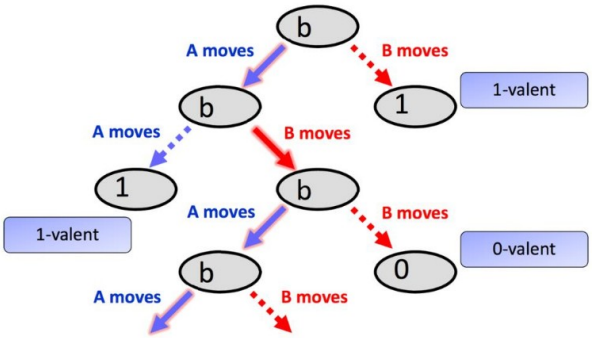
0valent: ניתן להגיע ממנו רק ל 0.

1 valent: ניתן להגיע ממנו רק ל 1.

טענה: קיים קלט למערכת שעבורו המצב הראשוני (השורש) הוא Bivalent.

קודקוד קריטי: הוא קודקוד bivalent שהילדים שלו אחד 0valent ואחד 1valent

Critical states



מתחילים במצב התחלתי bivalent. הפרוטוקול חייב להגיע למצב קריטי (critical state) אחרת היינו נשארים bivalent לנצח, והפרוטוקול לא ישיג הסכמה או סיום. המטרה היא כעת להשיג סתירה בכך שנראה שהמערכת יכולה להשאר bivalent לתמיד.

Reaching a critical state

המערכת יכולה להשאר bivalent לתמיד אם תמיד קיימת פעולה המונעת מהערכת להגיע למצב קריטי.

Model dependency

עד כה לא הסתמכנו על כך שלמודל יהיה זיכרון משותף.

וזה נכון ל: רג'יסטרים, message-passing, כל סוג של חישוב אסינכרוני.

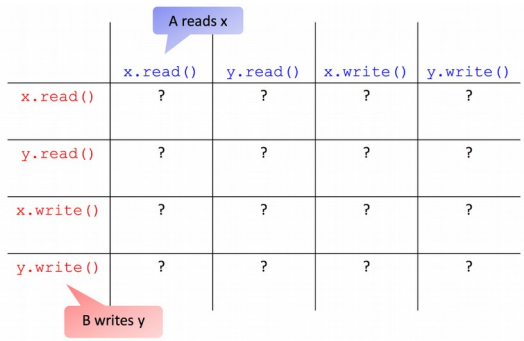
Steps with Shared Read/Write Registers:

תהליכים/טרדים.

מבצעים reads and/or writes.

לאותם אוגרים או לאוגרים שונים.

Possible interactions



ישנם 16 אפשרויות לקודקוד להיות קריטי.

נראה שעבור כל אחד מהמקרים הנ"ל בשלושה צעדים שקודקוד הוא לא קריטי.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

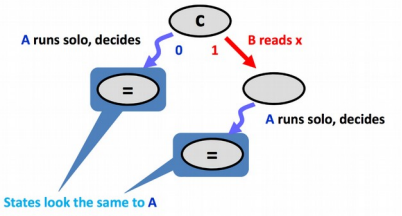
מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

אפשרות ראשונה קריאה מרג'יסטר:

בציור חלק מהעץ שבו אנו נראה כי קודקוד C אינו באמת קודקוד קריטי. אם C הוא קודקוד קריטי אז בהכרח הוא קודקוד ביוולנטי. לפחות אחד מהתהליכים רוצה לקרוא מ X, בה"כ B קורא מ X. נבחר את הענף שבו A רץ סולו, כנ"ל לאחר ש B קורה מ X, קודוקוד A רץ סולו. A כלל אינו יודע כי B לפני כן קרא את X, ולכן A מחליט את אותה החלטה.

ולכן נשארנו עם המקרים הבאים ---------->

כלומר בכל קודוקוד שבו לפחות אחד מאיתנו רוצה לקרוא הקודקוד אינו קריטי.



	x.read()	y.read()	x.write()	y.write()
x.read()	no	no	no	no
y.read()	no	no	no	no
x.write()	no	no	?	?
y.write()	no	no	?	?

אפשרות שניה כתיבה לרג'יסטרים שונים:

שוב, תחילה נניח בשלילה כי קודקוד C הוא קודקוד קריטי.

A רוצה לכתוב ל Y ו B רוצה לכתוב ל X.

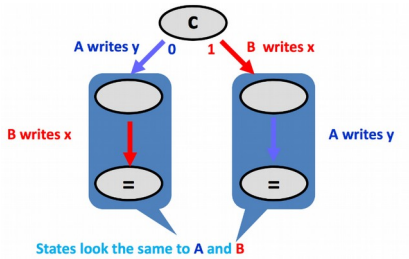
במידה ו A קודם כותב ל Y אז עדיין B רוצה לכתוב ל X.

במידה ו B קודם כותב ל X אז עדיין A רוצה לכתוב ל Y.

ולכן אלה הפעולות שבאות מייד אחרי. מכיוון שאלה תאים שונים A ו B לא יודעים איזו פעולה קרתה קודם. במקרה הזה הקודקודים נראים אותו הדבר עבור שני התהליכים אבל הענפים לא זהים.

ולכן גם במצב הזה הקודקוד אינו קודקוד קריטי ------->

נשאר עם המצב שבו שני התהליכים רוצים לכתוב לאותו הקודקוד.



	x.read()	y.read()	x.write()	y.write()
x.read()	no	no	no	no
y.read()	no	no	no	no
x.write()	no	no	?	no
y.write()	no	no	no	?

כתיבה לאותו הקודקוד:

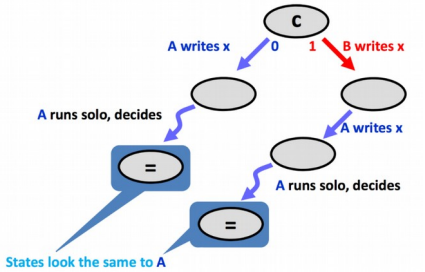
A רוצה לכתוב ל X מוביל ל 0valent

B רוצה לכתוב ל X מוביל ל 1valent

אם A כותב 0 ל X, נקח לאחר מכן את הענף שבו A רץ סולו.

אם B כותב 1 ל X, תהליך A עדיין רוצה לכתוב ל X, ניתן לו (למעשה הוא דורס את הערך ש B כתב), ולאחר מכן A רץ סולו.

סתירה לכן ש C הוא קריטי.



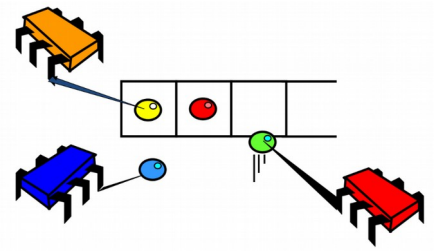
כלומר לא חשוב איזה אלגוריתם יש לנו, לא ניתר לפתור את בעיית הקונצנזוס תחת ההנחות (או קריאה או כתיבה באופן אטומי).

הסבר ההוכחה: הנחנו בשלילה שקיים כזה אלגוריתם ולכן קיים קודקוד קריטי (חייב להיות אחד כזה כי אחרת האלגוריתם לא מסתיים), בהוכחה בחנו את כל האפשרויות והראנו כי לא קיים קודקוד קריטי, סתירה להנחה שקיים אלגוריתם שכזה.

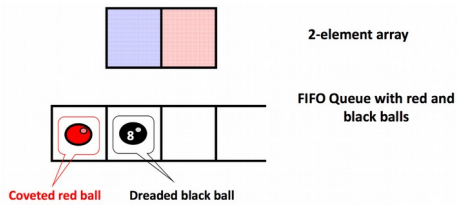
	x.read()	y.read()	x.write()	y.write()
x.read()	no	no	no	no
y.read()	no	no	no	no
x.write()	no	no	no	no
y.write()	no	no	no	no

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor
מבוסס על ההרצאות והמצגות של פרופ' דן חפץ



אנו רוצים לבנות תור FIFO.
לשם הפשטות נקח מקרה פרטי קטן בו יש רק 2 תהליכים, ואנחנו רק רוצים שהתהליכים יוציאו איברים מהתור תוך כדי שמירה על עקרונות ה FIFO.
נניח שקיים אלגוריתם שכזה, נקח אותו כקופסא שחורה.
יש 2 תאים בזיכרון שהתהליכים תחילה כותבים אליהם, בנוסף יש את תור ה FIFO מאותחל ב 2 כדורים, תחילה אדום ולאחר מכן שחור. 2 התהליכים הולכים לתור ולוקחים כדורים מהתור. אם התהליך הוציא כדור אדום, הוא יודע שהוא הראשון ולכן פולט את המספר שלו(לפי התא שלו בזיכרון). אם התהליך הוציא כדור שחור, אז התהליך ניגש לתא של התהליך השני והערך שרשום בתא של התהליך השני יהיה הפלט.



מבחינה טכנית התהליך הזה עובד, אבל הוכחנו מקודם שלא ניתן לפתור את בעיית הקונצנזוס עם 2 תהליכים שיכולים לקרוא או לכתוב כפעולה אטומית.
סתירה לכך שקיים האלגוריתם אותו לקחנו כקופסא שחורה. (הוכחה ברדוקציה)
(אם יש לי תור FIFO אני מצליח לתור בעיית קונצנזוס אבל לא ניתן לפתור את בעיית הקונצנזוס תחת ההנחות האלה –> לא ניתן לבנות תור FIFO שכזה תחת ההנחות).

Consensus 3: Read-Modify-Write Memory

ישנם n>= 2 מעבדים. אנחנו רוצים יישום שהוא wait-free.
תהליכים יכולים לקרוא ולכתוב לתא זיכרון משותף בפעולה אטומית אחת, כלומר, הערך שנכתב יכול להיות תלוי בערך הקריאה.
זה נקרא register (RMW) read-modify-write.
האם בעיית הקונצנזוס פתירה על ידי RMW ??

Consensus Protocol Using a RMW Register

A designated memory cell c is initialized to "?".

Every processor i runs the following algorithm:

r = read(c);

If r = ? **then** write(c, x_i) and decide x_i;

Else decide r.

The read and write operations above form one atomic step.

פעולת הקריאה והכתיבה באלגוריתם הנ"ל הוא פעולה אטומית. האם האלגוריתם הזה נכון?
האלגוריתם פועל נכון. מעבד אחד ניגש לראשונה ל C, מעבד זה יקבל את ההחלטה עבור המעבדים האחרים שיבוא אחריו. הפרוטוקול הוא ללא המתנה.

ראינו כי ניתן להשיג קונצנזוס באמצעות RMW רג'יסטרים, האם ניתן להשיג זאת גם עבור מודלים חלשים יותר?

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

RMW באופן פורמאלי:
כלומר תהיי F כלשהי, נרצה קודם לקרוא ואז להחליט.

```
public class RMW {
  private int value;

  public synchronized int rmw(function f) {
    int prior  = this.value;
    this.value = f(this.value);
    return prior;
  }

}
```

Return prior value

Apply function

בשיטה זו נוכל גם לעשות read()

```
public class RMW {
  private int value;

  public synchronized int read() {
    int prior  = this.value;
    this.value = this.value;
    return prior;
  }

}
```

Identify function

TAS :test and set
כל מה שהיא עושה היא מרימה flag.
קוראים מה שרשום, כותבים 1, מחזירים את מה שקראנו.

```
public class RMW {
  private int value;

  public synchronized int TAS() {
    int prior  = this.value;
    this.value = 1;
    return prior;
  }

}
```

Constant function

Fatch&Increment
יודעים כמה תהליכים היו שם לפני.

```
public class RMW {
  private int value;

  public synchronized int FAI() {
    int prior  = this.value;
    this.value = this.value+1;
    return prior;
  }

}
```

Increment function

Fatch & add
מוסיפים כל אחד כמה שהוא רואה(+X)

```
public class RMW {
  private int value;

  public synchronized int FAA(int x) {
    int prior  = this.value;
    this.value = this.value+x;
    return prior;
  }

}
```

Addition function

Swap
מעין פעולת כתיבה

```
public class RMW {
  private int value;

  public synchronized int swap(int x) {
    int prior  = this.value;
    this.value = x;
    return prior;
  }

}
```

Set to x

Compare & swap
2 קלטים old ו new אם היה כתוב את ה old , נחליף אותו ב new

```
public class RMW {
  private int value;

  public synchronized int CAS(int old, int new) {
    int prior  = this.value;
    if(this.value == old)
      this.value = new;
    return prior;
  }

}
```

"Complex" function

כולן פונקציות שבאמת יושמו ברמת החומרה. נבדוק האם ניתן יהיה לפתור בעזרתן את בעיית הקונצנזוס ?

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Definition of Consensus Number

לאובייקט יש מספר קונצנזוס n אם ניתן להשתמש בו יחד עם אוגרי קריאה/כתיבה אטומיים ליישום עם n תהליכים, אבל לא ניתן ליישמו עבור n+ 1 תהליכים.

דוגמה: אוגרי קריאה/כתיבה אטומיים יש להם קונצנזוס 1 (ניתן להשיג הסכמה עבור תהליך 1 אבל לא יותר).

Consensus Number Theorem

משפט: אם ניתן ליישם X מ Y ול X יש מספר קונצנזוס m, אז מספר הקונצנזוס של Y הוא לפחות m.

מספרי ההסכמה הם דרך שימושית לבדוק את רמת הסינכרון.

ניסוח אלטרנטיבי: אם ל X יש מספר קונצנזוס m, ול Y יש מספר קונצנזוס n<m אז אין דרך לבנות יישום ללא המתנה של X מ Y.

Consensus number of RMW

RMW הוא אינו טריוואלי אם קיים ערך x שעבורו: $x \neq f(x)$.

הפונקציות Test&Set, Fetch&Inc, Fetch&Add, Swap, Compare&Swap כלליות

הפונקציה read טריוויאלית.

משפט: לכל אובייקט RMW שאינו טריוויאלי יש מספר הסכמה לפחות 2.

הוכחה:

```
public class RMWConsensusFor2 implements Consensus{
  private RMW r;
  public Object decide() {
    int i  = Thread.myIndex();
    if(r.rmw(f) == v)
      return this.announce[i];
    else
      return this.announce[1-i];
  }
}
```

Initialized to v

Am I first?

Yes, return my input

No, return other's input

Interfering RMW

Let F be a family of functions such that for all f_i and f_j in F,

either They commute: $f_i(f_j(x))=f_j(f_i(x))$

Or they overwrite: $f_i(f_j(x))=f_i(x)$

טענה: לכל קבוצה כזו של אובייקטים של RMW יש מספר הסכמה לכל היותר 2.

לדוגמה:

Overwrite: Test&Set, Swap, Write.

Commute: Fetch&Inc, Fetch&Add, Read

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Consensus with Compare & swap

```
public class RMWConsensus implements Consensus {
  private RMW r;

  public Object decide() {
    int i = Thread.myIndex();
    int j = r.CAS(-1,i);
    if(j == -1)
      return this.announce[i];
    else
      return this.announce[j];
  }
}
```

Initialized to -1

Am I first?

Yes, return my input

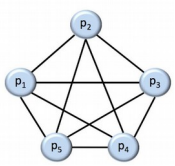
No, return other's input

כלומר עבור הפונקציה הנ"ל ניתן לפתור את בעיית הקונצנזוס עבור אינסוף תהליכים.

לסיכום, היררכיית הקונצנזוסים:

1	2	...	∞
• Read/Write Registers	• Test&Set • Fetch&Inc • Fetch&Add • Swap		• CAS • LL/SC

Consensus 4: Synchronous Systems

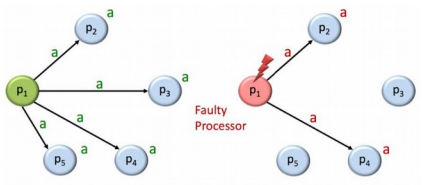


נדבר על מערכת סינכרונית בגרף שלם, כלומר, כל אחד יכול לשלוח הודעה לכל אחד אחר, יש סיבובים.

Broadcast: שלח הודעה לכל הצמתים (המעבדים) בסיבוב אחד.

בסוף הסיבוב כולם מקבלים את ההודעה.

כל צומת יכול לשדר ערך בכל סיבוב.



Crash failures: שידור יכול להכשל אם צומת קרס.

לדוגמה: חלק מההודעות עלולות ללכת לאיבוד.

ב crash failures הקודקוד שנכשל לא חוזר לפתע לפעול בהמשך.

Recalling consensensus

קלט: לכולם יש ערך התחלתי.

הסכמה: על כולם להחליט על אותו ערך.

Validity: אם כולם מתחילים באותו ערך, אז כולם צרכים להחליט את הערך הזה.

סיום: כל תהליך צריך להסתיים לאחר מספר סופי של סיבובים.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

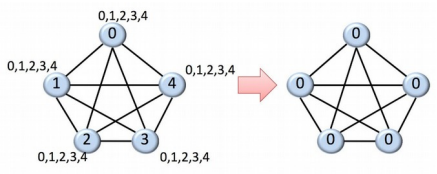
מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

אלגוריתם קונצנזוס פשוט

כל תהליך מבצע את הפעולות הבאות:

- תעשה broadcast על ערך הקלט שלך.
- החלט על המינימום של כל הערכים שהתקבלו (כולל שלך).

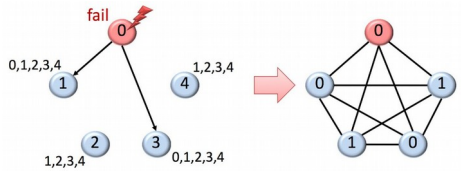
הערה: האלגוריתם הזה דורש סיבוב אחד בלבד. בפרט מתקיים סיום.



בביצוע ללא כישלונות:

שידור הערכים והחלטה על המינימום משיגים הסכמה.

מושג גם validity: אם כולם מתחילים עם אותו ערך התחלתי, כולם מסכימים על הערך הזה מכיוון שהוא המינימום.



בביצוע עם כשלונות:

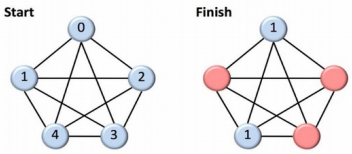
הערך של התהליך שנכשל אינו מגיע לחלק מהתהליכים האחרים.

במקרה זה, בחירת המינימום עשויה לא להוביל להסכמה.

m-Resilient Consensus Algorithm

אם אלגוריתם פותר הסכמה כל עוד יש מקסימום m תהליכים שנכשלו, אז אנו אומרים שזה אלגוריתם קונסנזוס m-resilient.

דוגמה: הקלט ופלט של אלגוריתם קונצנזוס 3-resilient



הסכמה: כל התהליכים מסכימים על אחד מהקלטים.

An m-Resilient Consensus Algorithm

כל תהליך מבצע את הפעולות הבאות:

סיבוב 1: שדר לכולם את ערך הקלט שלך.

סיבוב 2 עד m+1: שדר את המינימום של הערכים שהתקבלו אלא אם הוא נשלח כבר לפני כן.

סוף הסיבוב ה m+1: החליטו על הערך המינימלי שהתקבל.

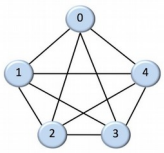
סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

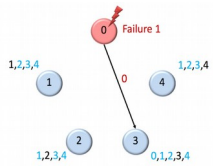
מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

דוגמה: מספר הכשלים המותרים m= 2 , ולכן צריך m+ 1= 3 סיבובים.

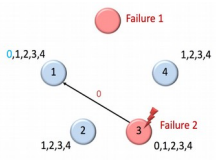
סיבוב 0:



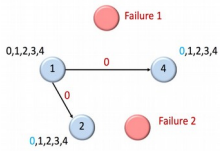
סיבוב 1: כל תהליך שידר את הקלט שלו לכל שאר התהליכים –>



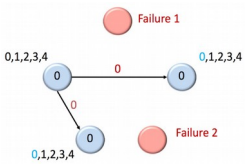
סיבוב 2: כל תהליך משדר את הערך המינימלי שקיבל לכל שאר התהליכים



סיבוב 3: כל תהליך משדר את הערך המינימלי שקיבל לכל שאר התהליכים



סוף סיבוב 3: כל תהליך מחליט את המינימום מבין הערכים שקיבל



ניתוח:

בסוף הסיבוב בו אף תהליך אינו נכשל, כל תהליך (שאינו נכשל) יודע את כל הערכים של כל התהליכים האחרים המשתתפים.

ידע זה לא משתנה עד סוף האלגוריתם. לכן, כל התהליכים יחליטו על אותו הערך.

עם זאת, מכיוון שאנו לא יודעים את האינדקס של הסיבוב שבו זה יקרה, עלינו לאפשר לאלגוריתם לבצע m+ 1 סיבובים.

Validity: אם כל התהליכים מתחילים באותו עם אותם קלטים, הם יחליטו על אותו הערך.

Lower bound on the number of rounds

משפט: במערכת העברת הודעות סינכרונית עם $n \geq f+2$ צמתים, כל אלגוריתם f–resilient דורש לפחות $f+1$ סיבובים.

הערה: לפני הוכחת המשפט אנו רואים "תרחיש המקרה הגרוע ביותר": בכל סיבוב אחד התהליכים נכשל.

הוכחה: במצגות.

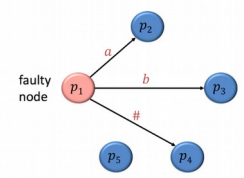
Arbitrary Behaviour

ההנחה שתהליך התרסק ומפסיק לפעול לנצח, לפעמים אופטימית מידי. אולי תהליך התרסק ולאחר מכן התאושש.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor
מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Consensus 5: Byzantine Failures



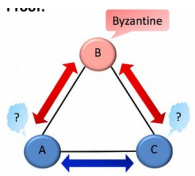
תהליכים שונים עשויים לקבל ערכים שונים. תהליך ביזנטי יכול להתנהג כמו crash–failed תהליך. לאחר הכישלון, הצומת (תהליך) נשאר ברשת.

ננסה להגיע להסכמה גם עם Byzantine Failures:
כמו בעבר, אם אלגוריתם פותר קונצנזוס עבור m תהליכים כושלים, אנו אומרים שהוא אלגוריתם הסכמה m–resilient.
Validity: אם כל התהליכים הלא לקויים מתחילים באותו ערך, אז כל התהליכים הלא לקויים מחליטים על אותו ערך.
נשים לב שבאופן כללי Validity אינו מבטיח כי הערך הסופי הוא ערך קלט של תהליך שאינו ביזנטי.
עם זאת, אם הקלט הוא בינארי, אז מ Validity נדע שתהליכים יחליטו על ערך שלפחות אחד תהליך לא–ביזנטי.
ברור כי כל אלגוריתם הפותר קונצנזוס m–resilient דורש לפחות m+ 1 סיבובים (נובע מההוכחה על גבול תחתון של crash–failure.

Impossibility

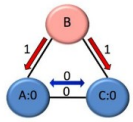
משפט: אין אלגוריתם m–resilient הפותר הסכמה במערכת העברת הודעות סינכרונית עם צמתים עם תקלות ביזנטיות, עבור כל $m \geq n/3$.
מתווה ההוכחה: ראשית נוכיח את המקרה עבור n= 3 שהוא לא אפשרי עבור m= 1. לאחר מכן ניתן להוכיח את המקרה הכללי על 3 הצמיתם באמצעות רדוקציה.

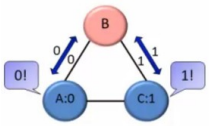
Impossibility for 3 nodes and 1 failure



למה: אין אלגוריתם resilient-1 לשלושה צמתים (המשיג קונצנזוס).

סקיצה של ההוכחה:
נניח של A וגם ל C יש קלט 0.
אם הם מחליטים על 1, validity מופר.
מכאן שעל A ו C להחליט 0 ללא תלות ב B.
באופן דומה, A ו C חייבים להחליט 1 אם הקלטים שלהם יהיו 1.
אם הקלט של A הוא 0 ו– B אומר ל– A שהקלט שלו הוא 0, A מחליט על 0.
אם הקלט של C הוא 1 ו– B אומר ל– C שהקלט שלו הוא 1, C מחליטה על 1.



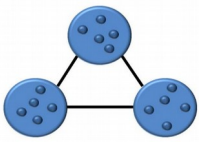


סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

המקרה הכללי:



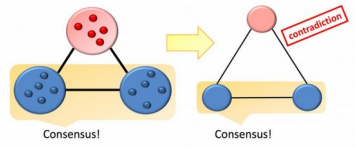
נניח בשלילה שקיים אלגוריתם m–resilient עבור n קודקודים כאשר $m \geq n/3$.

נשתמש באלגוריתם זה כדי לפתור הסכמה עבור 3 צמתים עם צומת ביזנטי אחד.

לשם הפשטות נניח כי n מתחלק ב 3.

ניתן לכל אחד משלושת הצמתים לדמות n/3 תהליכים.

אחד מבין שלושת התהליכים הוא ביזנטי –> ה n/3 הצמתים המדומים שלה עשויים להתנהג כצמתים ביזנטיים בעוד שהאלגוריתם סובל מ n/3 קודקודים ביזנתיים, והוא עדיין יכול להגיע לקונצנזוס. כלומר, פתרנו את בעיית הקונצנזוס עבור שלושה תהליכים.

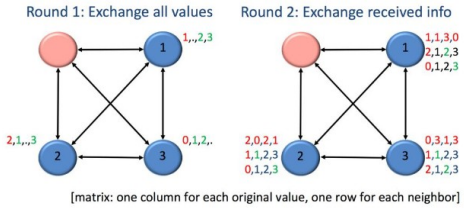


Consensus 6: A Byzantine Agreement Algorithm

האם הצמתים יכולים להגיע לקונצנזוס עם $n > 3m$?

שאלה פשוטה יותר: מה אם $n = 4$ ו– $m = 1$?

התשובה היא כן, וזה לוקח שני סיבובים:



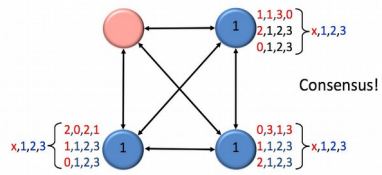
אלגוריתם:

לאחר הסיבוב השני כל צומת קיבל 12 ערכים, 3 עבור כל אחד מארבעת ערכי הקלט (העמודות). אם לפחות 2 מתוך 3 ערכי העמודה שווים, ערך זה מתקבל, אחרת הוא לא מתקבל.

ערכים של צמתים לא תקולים מתקבלים.

הערה: במצגת בשלב השני הווקטור העצמי אינו מופיע זאת טעות, הוא אמור להופיע כשורה במטריצה. כלומר לכל צומת יהיה 16 ערכים ולא 12.

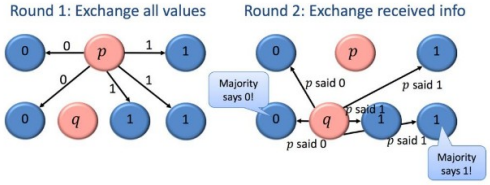
הערך של הצומת הביזנטי מתקבל אם ורק אם הוא שולח את אותו ערך לפחות לשני צמתים בסיבוב הראשון.



הצמתים מחליטים על הערך שהתקבל מספר הפעמים הגדול ביותר (במידה ויש יותר מאחד כזה– ניקח את המינימום).

האלגוריתם:

- כולם שולחים קלט.
- כולם שולחים את הווקטור שהורכב מבקלטים בסיבוב הקודם.
- כולם מחזיקים כעת מטריצה, מוחקים את האלכסון במטריצה, מכל עמודה לוקחים את המספר שהופיע הכי הרבה, אם אין כזה מתעלמים.
- מהשלב הקודם נקבל n מספרים, הפלט יהיה המספר שמופיע הכי הרבה, אם אין נחליט על המינימלי.



האם האלגוריתם הנ"ל עובד באופן כללי עבור כל m and n $3m >$?

התשובה היא לא. ננסה עבור m= 2 and n= 7:

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

הבעיה היא ש q יכול להגיד דברים אחרים מאשר מה ש p באמת שלח ל q.
מה הפתרון לבעיה הזו ?

הפתרון פשוט: החלף את כל המידע פעם נוספת.
באופן זה, התהליכים יכולים ללמוד ש q מסר מידע לא עקבי על p. מכאן ניתן לשלול את אותם הצמתים שאינם מוסרים מידע עקבי.
אם $m = 2$ ו- $n > 6$, ניתן להגיע לקונצנזוס בשלושה סיבובים.
למעשה, האלגוריתם הבא פותר את הבעיה עבור כל m וכל $n > 3m$:
החלף את כל המידע במשך $m + 1$ סיבובים.
התעלם מכל התהליכים שסיפקו מידע לא עקבי.
תן לכל התהליכים להחליט על סמך אותו קלט.

לאלגוריתם הנ"ל מספר יתרונות:

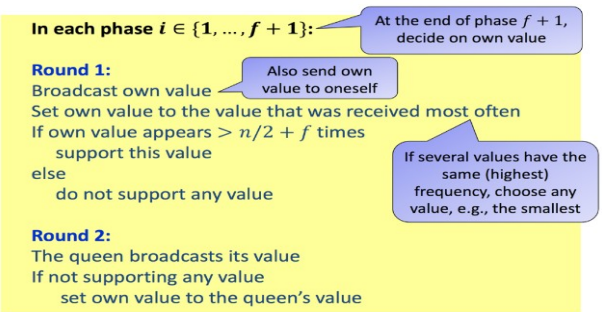
- האלגוריתם עובד עבור כל m ו- $n > 3m$, וזה האופטימלי.
- האלגוריתם לוקח רק סיבוב $m + 1$. זה אפילו אופטימלי לכישלונות התרסקות.
- האלגוריתם עובד לכל קלט ולא רק לקלט בינארי.

עם זאת, יש לאלגוריתם כמה חסרונות ניכרים:

- "להתעלם מכל התהליכים שסיפקו מידע לא עקבי" קשה לפרמול.
- גודל ההודעות גדל באופן אקספוננציאלי. וזו בעיה קשה. לכן כדאי ללמוד לבדוק האם ניתן לפתור את הבעיה באמצעות הודעות קטנות יותר.

Consensus 7: The Queen Algorithm

אלגוריתם המלכה הוא אלגוריתם הסכמה ביזנטי פשוט המשתמש בהודעות קטנות.
אלגוריתם המלכה פותר הסכמה עם n צמתים f כשלים ב $f + 1$ שלבים בתנאי ש $n > 4f$.
רעיון מרכזי:
יש מלכה שונה (ידועה מההתחלה) בכל שלב.
מכיוון שיש שלבים $f + 1$ (לכל שלב 2 סיבובים), קיים שלב בו המלכה אינה ביזנטית (שובך היונים).וודא שבשלב זה כל הצמתים בוחרים באותו ערך ושבשלבים העתידיים הצמתים לא ישנים יותר את הערכים שלהם.
האלגוריתם:

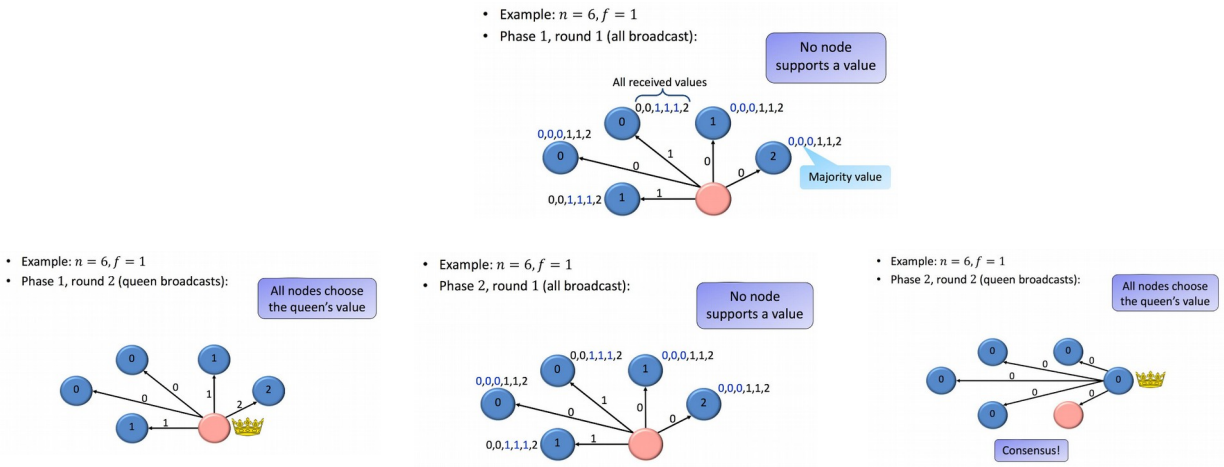


סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

דוגמת הרצה:



ניתוח:

אחרי השלב בו המלכה לא ביזנטית, לכל הצמתים הלא ביזנטיים יש אותו ערך:

אם כל הצמתים משנים את ערכיהם לערך המלכה, ברור שכל הערכים זהים.

אם צומת כלשהו לא משנה את ערכו לערך המלכה, הוא קיבל את הערך יותר מ- $n/2 + f$ פעמים, אזי כל הצמתים הלא ביזנטיים האחרים (כולל המלכה) קיבלו ערך זה יותר מ- $n / 2$ פעמים וכך כל הצמתים הלא ביזנטיים שיתפו את הערך הזה.

בכל השלבים העתידיים, אף צומת לא משנה את ערכו:

בסיבוב הראשון של שלב כזה, צמתים מקבלים את הערך שלהם לפחות מ- $n-f > n/2 + f$ צמתים ולכן אינם משנים את ערכם.

למעשה, כל הצמתים הלא ביזנטיים תומכים בערך זה ובכך אינם מקבלים את הצעת המלכה בסיבוב השני.

לפיכך, כל הצמתים הנכונים תומכים באותו ערך. זה גם מוכיח validity.

סיכום אלגוריתם המלכה:

לאלגוריתם של המלכה מספר יתרונות:

- ההודעות קטנות והצמתים מחליפים את הערכים הנוכחיים שלהם.
- האלגוריתם עובד לכל קלט ולא רק לקלט בינארי.

עם זאת, יש לו גם כמה חסרונות:

- האלגוריתם דורש שלבים $f + 1$ המורכבים משני סיבובים כל אחד. כפול מהאלגוריתם האופטימלי.
- האלגוריתם עובד רק עם $f < n/4$ צמתים ביזנטיים.

האם ניתן לתכנן אלגוריתם שעובד עם $f < n/3$ צמתים ביזנטיים שמשתמש בהודעות קטנות?

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

Consensus 8: The King Algorithm

אלגוריתם המלך הוא אלגוריתם המצליח להתמודד עם $f < n/3$ כשלים ביזנטיים ומשתמש בהודעות קטנות.

קינג אלגוריתם דורש $f + 1$ שלבים (כאן כל שלב מורכב משלושה סיבובים).

רעיון מרכזי:

הרעיון הבסיסי זהה לאלגוריתם של המלכה.

יש מלך שונה (ידוע מההתחלה) בכל שלב.

מכיוון שיש שלבים $f+1$, קיים שלב בו המלך אינו ביזנטי. (שובך היונים).

ההבדל העיקרי לאלגוריתם של המלכה הוא שישנו שלב ביניים בו הצמתים מציעים ערך אם הם קיבלו אותו פעמים רבות. ערך מוצע מתקבל אם צמתים רבים מציעים אותו.

האלגוריתם:

In each phase $i \in \{1 \dots f+1\}$:

At the end of phase $f+1$, decide on own value

Round 1:

Broadcast own value

Also send own value to oneself

Round 2:

If some value x appears $\geq n - f$ times

Broadcast "Propose x"

If some proposal received $> f$ times

Set own value to this proposal

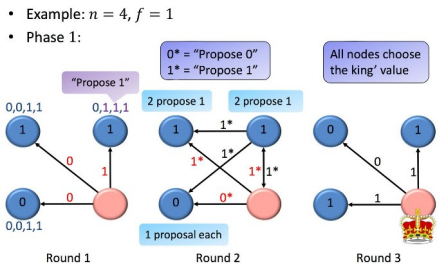
Round 3:

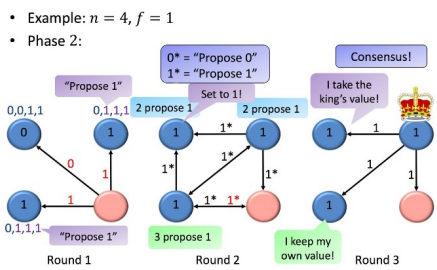
The king broadcasts its value

If own value received $< n - f$ proposals

Set own value to the king's value

הרצה לדוגמה:





הסבר אלגוריתם:

דרישה $n > 3f$, כל שלב באלגוריתם מתחלק לשלושה סיבובים, בכל שלב מלך אחר, לאלגוריתם f+1 סיבובים.

סיבוב ראשון: כל הקודקודים שולחים ערך עצמי.

סיבוב שני: אם בסיבוב הראשון, ערך כלשהו X התקבל n-f פעמים או יותר, שלח "אני מציע X".

אם הצעה כלשהי התקבלה מעל ל f פעמים קח ערך זה.

סיבוב שלישי: המלך משדר ערך עצמי, כל מי שהערך שהוא מחזיק התקבל מתחת ל n-f פעמים מקשיב למלך.

לאחר f+1 שלבים פלוט ערך עצמי.

סיכום קורס מערכות מבוזרות

נכתב על ידי: שי נאור https://github.com/shaynaor

מבוסס על ההרצאות והמצגות של פרופ' דן חפץ

האלגוריתם של המלך: ניתוח

הבחנה: אם הצומת הנכון כלשהו מציע x, אף צומת לא ביזנטי אחר לא מציע שום $y \neq x$.
שני הצמתים יצטרכו לקבל את אותו ערך לפחות $n\text{-}f$ פעמים, כלומר, שני הצמתים קיבלו את ערכם לפחות מ $n\text{-}2f$ צמתים לא ביזנטיים. בסך הכל חייבים להיות לפחות $f > n + (n\text{-}2f)2$ צמתים, סתירה.

validity: אם כל הצמתים הלא ביזנטיים מתחילים באותו ערך, כל הצמתים הלא ביזנטיים מקבלים ערך זה לפחות $n\text{-}f$ פעמים ואז מציעים אותו. כל הצמתים הלא ביזנטיים מקבלים הצעות לערך שלהם לפחות $n\text{-}f$ פעמים, כלומר, אף צומת לא ביזנטי לא ישנה את ערכו לערך המלך.

לאחר השלב בו המלך לא ביזנטי, לכל הצמתים הלא ביזנטיים יש אותו ערך.
אם כל הצמתים משנים את ערכיהם לערך המלך, ברור שכל הערכים זהים.
אם צומת כלשהו לא משנה את ערכו לערך המלך, הוא בטח קיבל הצעה לפחות $n\text{-}f$ פעמים.
כלומר, לפחות $n\text{-}2f$ צמתים לא ביזנטיים משדרים הצעה זו וכל התהליכים הלא ביזנטיים מקבלים אותה לפחות $n\text{-}2f > f$ פעמים. מכאן שכל הצמתים הלא ביזנטיים קובעים את ערכם לערך המוצע.
בכל השלבים העתידיים, אף צומת לא משנה את ערכו.
זה נובע מיד מהעובדה שלכל הצמתים הלא ביזנטיים יש ערך זהה לאחר השלב בו המלך היה ומתוקף validity.

האלגוריתם של המלך: סיכום

לאלגוריתם של המלך כמה יתרונות:

- האלגוריתם עובד עבור כל f ו- $n > 3f$, וזה האופטימלי.
- ההודעות הן קטנות, צמתים רק מחליפים את הערכים הנוכחיים שלהם.
- האלגוריתם עובד לכל קלט ולא רק לקלט בינארי.

עם זאת, יש לאלגוריתם גם חסרון:

- האלגוריתם דורש $f + 1$ שלבים המורכבים משלושה סיבובים כל אחד. זה פי שלושה מהאלגוריתם האופטימלי.