אילן הינדי

[הקלד את תקציר המסמך כאן. התקציר הוא בדרך כלל סיכום קצר של תוכן המסמך. הקלד את תקציר המסמך כאן. התקציר הוא בדרך כלל סיכום קצר של תוכן המסמך.]

אלגוריתמים מבוזרים

פרק 3 : Snapshots

פרק 3 : Snapshots

תוכן עניינים

[1 הגדרות 2](#_Toc7438863)

[2 שימושים 2](#_Toc7438864)

[3 בעיות 2](#_Toc7438865)

[4 תנאים ל-consistent snapshot 3](#_Toc7438866)

[5 אלגוריתם Chandy-Lamport 3](#_Toc7438867)

[5.1 עקרונות 3](#_Toc7438868)

[5.2 Pseudo code 4](#_Toc7438869)

[5.3 דוגמא 6](#_Toc7438870)

[5.4 תוספת למימוש : דווח של ה – snapshot לתהליך היוזם 6](#_Toc7438871)

[5.4.1 כללי 6](#_Toc7438872)

[5.4.2 מבני נתונים והודעות 7](#_Toc7438873)

[5.4.3 מהלך האלגוריתם 7](#_Toc7438874)

פרק 3 : Snapshots

# הגדרות

Snapshot – מצב המשתנים הלוקליים בכל תהליך וההודעות בערוצים ב-configuration מסוים.

1. Basic message – הודעה של התהליך המרכזי
2. Control message- הודעה של תהליך ה-snapshot
3. Consistent snapshot- snapshot שמתאר קונפיגורציה של התוכנית (כלומר המצב שמתואר ב-snapshot מתואר גם בדיאגרמת המצבים של התוכנית)
4. Inconsistent snapshot – snapshot שמתאר קונפיגורציה שאינה קיימת של התוכנית (כלומר המצב שמתואר ב-snapshot לא מתואר בדיאגרמת המצבים של התוכנית)
5. Presnapshot- אירוע שקרה בתהליך לפני snapshot מקומי
6. Postsnapshot- אירוע שקרה בתהליך אחרי snapshot מקומי

# שימושים

לבחינת תכונות חיצוניות לתהליכים שנשארות נכונות מהרגע שהן נעשות נכונות לדוגמא:

* Deadlock
* Termination
* Garbage
* Restart after failure
* Debugging

# בעיות

נניח שתהליך מעונין לקבל את הסטטוס של הביצוע של אלגוריתם מבוזר הוא צריך לבקש מכל שאר התהליכים לקחת Snapshot מקומי. בנוסף תהליכים צריכים לחשב את המצב בערוצים של הודעות במעבר ברגע הלוגי של ה – snapshot.

הסיבוך הוא שתהליכים לוקחים snapshot מקומי בזמנים שונים לכן snapshot לא בהכרח מציין קונפיגורציה של הביצוע.

1. להבטיח שה-snapshot יהיה של configuration חוקי. כלומר שלא יוצר מצב שבגלל שהחישוב של ה-snapshot מופץ ברשת ולא מתרחש באותו זמן בכל התהליכים לא יקרה מצב שה-snapshots המקומיים לא מרכיבים קונפיגורציה לא קיימת
2. לפתח אלגוריתם שלא יפריע לאלגוריתם שעובד
3. לפתח אלגורית שלא דורש הקפאה של התהליך המרכזי
4. נסתכל על המצב הבא בערוץ PQ:
   1. בתהליך השולח אירוע השליחה postsnapshot כלומר הסדר היה לקיחת snapshot ואח"כ שליחת ההודעה
   2. בתהליך המקבל אירוע הקבלה הוא presnapshot כלומר הסדר היה קודם קבלת ההודעה וא"כ לקיחת snapshot
   3. זה מצב לא הגיוני מאחר ומקבלים סדר כזה : קבלת ההודעה – ביצוע snapshot – שליחת ההודעה שזה סדר לא הגיוני.
5. נסתכל על מצב בערוץ PQ
   1. בתהליך השולח אירוע השליחה הוא presnapshot כלומר הודעה נשלחת לפני לקיחת ה - snapshot
   2. בתהליך המקבל אירוע הקבלה הוא postsnapshot כלומר ההודעה התקבלה אחרי ה-snapshot
   3. זה מצב לא הגיוני מאחר ומקבלים סדר כזה שליחת ההודעה – ביצוע ה – snapshot – קבלת ההודעה (יש להם פה איזה אי הבנה בספר מצב כזה הוא מצב שיש הודעה בערוץ יכול להיות שהם מתכוונים לכך שזה לא חוקי שיש הודעה בערוץ ואז אין קונפיגורציה חוקית ????)

# תנאים ל-consistent snapshot

1. כל אירוע שהוא presnapshot a מקיים שכל אירוע b שמקיים ) b ≺ aכלומר b לפני a ב-casual order) הוא גם presnapshot. תנאי זה מבטיח שהפרש בזמני הדיווח של התהליכים לא יגרום לכך שתהליך אחד "יתקדם" מעבר לתהליך אחר.
2. הודעה נכללת בערוץ ב-snapshot אם ורק אם בתהליך ששולח היא presnapshot ובתהליך המקבל היא postsnapshot

# אלגוריתם Chandy-Lamport

## עקרונות

1. אחת התחנות יוזמת snapshot גלובלי
2. התחנה מבצעת snapshot מקומי ושולחת control message(marker) לכל הערוצים שיוצאים ממנה
3. כל תחנה מבצעת snapshot מקומי כאשר היא מקבלת את ה-marker הראשון. ושולחת הודעות marker לכל השכנים שלה.
4. המצב בערוץ מחושב ע"י התחנה ככל ההודעות שהגיעו לתחנה מהערוץ מאז ה –snapshot שהיא ביצעה כאשר התקבל ה – marker הראשון.
5. כאשר תחנה מקבלת marker היא שולחת לכל הערוצים שיוצאים ממנה את ה-snapshot המקומי בתוספת דווח על ההודעות שהתקבלו בערוץ שממנו הגיע ה-marker מאז ה-snapshot המקומי
6. הערוצים הם FIFO
7. כאשר התהליך מקבל marker מכל הערוצים הנכנסים התהליך מפסיק

## Pseudo code

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Messages | | | | | | | | | | | | | | | | | | | | | | |
|  | **(m)** | | | | - | | base algorithm message | | | | | | | | | | | | | | |  |
|  | **(marker)** | | | | - | | snapshot algorithm message | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Variables | | | | | | | | | | | | | | | | | | | | | | |
|  | recordedP | | | | - | | true if P takes local snapshot | | | | | | | | | | | | | | |  |
|  | makerp[c] | | | | - | | true if the marker message arrived to P through channel c | | | | | | | | | | | | | | |  |
|  | stateP[c] | | | | - | | a stack holding the messages to P arrived from channel c after the local snapshot in P | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Initialize | | | | | | | | | | | | | | | | | | | | | | |
|  | recordedP | | | | ← | false | | | | | | | | | | | | | | | |  |
|  | makerp[c] | | | | ← | false for all channel c | | | | | | | | | | | | | | | |  |
|  | stateP[c] | | | | ← | ∅ for all channel c | | | | | | | | | | | | | | | |  |
|  | **if** P wants to initiate snapshot | | | | | | | | | | | | | | | | | | | | |  |
|  |  | perform procedure TakeSnapshot | | | | | | | | | | | | | | | | | | | |  |
|  | **end if** | | | | | | | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Algorithm | | | | | | | | | | | | | | | | | | | | | |  |
|  | **if P receives a basic message (m) through incoming channel c0** | | | | | | | | | | | | | | | | | | | | |  |
|  | **if** **recordedP** = true and **makerp[c]**= false | | | | | | | | | | | | | | | | | | | | |  |
|  |  | **stateP[c]** ← append(**stateP[c]**, **(m)**) | | | | | | | | | | | | | | | | | | | |  |
|  | **end if** | | | | | | | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **if P receives (marker) through incoming channel c0** | | | | | | | | | | | | | | | | | | | | |  |
|  | perform procedure TakeSnapshot | | | | | | | | | | | | | | | | | | | | |  |
|  | **makerp[c]**← true | | | | | | | | | | | | | | | | | | | | |  |
|  | **if** **makerp[c]** = true for all incoming channels | | | | | | | | | | | | | | | | | | | | |  |
|  |  | terminate | | | | | | | | | | | | | | | | | | | |  |
|  | **end if** | | | | | | | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **Procedure TakeSnapshot** | | | | | | | | | | | | | | | | | | | | |  |
|  | **if** **recordedP** = false | | | | | | | | | | | | | | | | | | | | |  |
|  |  | **recordedP** ← true | | | | | | | | | | | | | | | | | | | |  |
|  |  | send **(marker)** into each outgoing channel of **P** | | | | | | | | | | | | | | | | | | | |  |
|  |  | take local snapshot of the state of **P** | | | | | | | | | | | | | | | | | | | |  |
|  | **end if** | | | | | | | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## דוגמא



## תוספת למימוש : דווח של ה – snapshot לתהליך היוזם

### כללי

1. האלגוריתם הבא הוא גרסה של weight throwing algorithm לבדיקה האם כל הרשת סיימה.
2. בגרסה מצב סיום הוא מצב שבו תהליך קיבל (marker) מכל השכנים שלו ולפי האלגוריתם למציאת snapshot הוא עוצר
3. בדומה לתהליך למציאת termination לכל הודעת (marker) יתוסף משקל
4. כאשר התהליך היוזם מתחיל סבב הוא שם את המשקל שלו כ – 1
5. כאשר תהליך מקבל marker הוא מוסיף את המשקל של ההודעה למשקל שלו
6. כאשר תהליך רוצה לשלוח הודעת (marker) הוא מחלק את המשקל שלו לחצי ושולח עם כל הודעה את חצי המשקל שלו מחולק למספר היעדים של ההודעה
7. כאשר תהליך קיבל הודעות (marker) מכל השכנים שלו במקום לעצור הוא שולח הודעת תשובה שהיעד שלה הוא התהליך היוזם
8. כל תהליך שמקבל הודעת תשובה כזו מעביר אותה לכל השכנים שלו רק בתנאי שזו הפעם הראשונה שהוא קיבל הודעת תשובה מהתהליך הנ"ל
9. התהליך היוזם צובר את כל הסכומים שהתקבלו (פעם אחת מכל תהליך) וכאשר הסכום מגיע ל – 1 הוא יודע שהשלב הסתיים

### מבני נתונים והודעות

1. פרמטרים שמוחזקים בכל תהליך לטובת התהליך :
   1. waightP – משקל של התהליך שנצבר כאשר הוא מקבל הודעות (marker)
   2. ReceivedStapshotFrom – רשימה שמתאפסת כאשר מתחיל סבב וכוללת את כל התהליכים שמהם התקבלה הודעת SnapshotReport לסבב הנוכחי
2. פרמטרים שנוספים להודעת (marker) weight:
3. סוג הודעה נוסף (Report) – הודעה שבה מדווח ה – snapshot ההודעה כוללת את השדות הבאים:
   1. IdP : ה – id של התהליך המדווח
   2. snapshotP : ה – snapshot של התהליך המדווח
   3. WeightP : המשקל של התהליך המדווח כאשר הגיעו אליו הודעות (marker) מכל הערוצים

### מהלך האלגוריתם

1. כאשר התהליך היוזם מאתחל סבב בתהליך ה – snapshot הוא מאתחל weightinitiator ← 1
2. כאשר תהליך Pמקבל הודעת (marker, weightm):
   1. אם מדובר ב – (marker) הראשון בסבב (כלומר recordedP = false):
      1. weightP ← weightm
      2. ReceivedReportFrom← ∅
      3. לכל הודעה שנשלחת מוצמד משקל שהוא חצי מהמשקל שעל ההודעה מחולק במספר היעדים
   2. אחרת
      1. weightP ← weightP + weightm
   3. אם התקבלו הודעות מכל הערוצים הנכנסים
      1. הוסף את idP (ה – Id העצמי) לרשימה ReceivedReportFrom
      2. שלח הודעת (Report,IdP, snapshotP, weightP) לכל הערוצים היוצאים
3. כאשר תהליך P מקבל הודעה (Report, IdQ, snapshotQ, weightQ)
   1. אם IdQ נמצא ברשימה ReceivedReportFrom P זורק את ההודעה
   2. אחרת
      1. הוסף את IdQ לרשימה ReceivedReportFrom
      2. אם P אינו התהליך היוזם
         1. שלח את ההודעה בכל הערוצים היוצאים
      3. אחרת (אם P הוא התהליך היוזם)
         1. הצב : weightP ← weightP + weightQ
         2. שמור את snapshotQ
         3. אם weightp = 1 סיים