# מערכות מבוזרות - הקדמה

## הגדרה

מערכת מבוזרת היא אוסף של מכשירי מחשוב נפרדים היכולים לתקשר זה עם זה. לכל מעבד במערכת מבוזרת בדרך כלל יש את האג'נדה העצמאית שלו, אך מסיבות שונות כגון: צורך בשיתוף משאבים, העלאת זמינות וסובלנות לקריסות, על המעבדים לתאם את פעולותיהם. כיום, כמעט כל חישוב שנעשה הוא במערכות מבוזרות, הסיבה לכך היא שמערכות מבוזרות מאפשרות לנו:

* שיתוף נתונים.
* לטפל בכמויות מידע גדולות יותר, שכן אין צורך שכל יחידת מחשוב תחזיק את כל המידע אלא ניתן לפזר את המידע ביניהם או שכולם יפנו למקומות מוגדרים ששם מאוחסן המידע.
* חישוב מקבילי על פני מכונות רבות.
* לבנות תשתיות תקשורת ומערכות המשתרעות על פני מרחקים גדולים.
* לבנות מערכות שעמידות לתקלות, כך שאם יחידת מחשוב אחת או יותר יקרסו המערכת כולה עדיין תמשיך לתפקד.

הקשיים העיקריים במערכות מבוזרות הם: כיצד לחלק את העבודה בין יחידות המחשוב? כיצד לשתף חישוב, נתונים, ותשתיות תקשורת? כיצד מתמודדים עם הודעות שנאבדו או עיכובים בהודעות? וכיצד מתמודדים עם יחידות מחשוב שקרסו, נפרצו, או שעובדות באיטיות?

## מודלים תיאורטיים של מערכות מבוזרות

בקורס זה נתמקד בשני סוגי מודלים: העברת הודעות וזיכרון משותף. כל אחד מהם יכול להיות מסונכרן או לא מסונכרן. שני המודלים שקולים, כלומר כל מה שמודל אחד יכול לעשות גם המודל השני יכול, ולהיפך, אמנם ביעילויות שונות.

**העברת הודעות**: זהו מודל שבו כל יחידות המחשוב, הנקראות במודל זה צמתים, מתקשרות זו עם זו באמצעות הודעות. ניתן לייצג מערכת כזו בגרף שבו כל יחידת מחשוב היא צומת, וקיימת צלע בין כל שני צמתים שיכולים לתקשר ביניהם. מודל זה משמש למערכות גדולות או רשתות תקשורת.

**זיכרון משותף**: זהו מודל שבו כל יחידות המחשוב מתקשרות על ידי כתיבה/קריאה ממקום בזיכרון המשותף לכל יחידות המחשוב. מודל זה משמש בעיקר במקרים של מחשב יחיד שיש בו כמה מעבדים, או מעבד שרצים בו מספר תהליכים.

### מערכת מסונכרנת

זוהי מערכת שמחלקת את הזמן ליחידות זמן קבועות הנקראות rounds וממוספרות . כל round הוא בין הזמן ל-. כל התהליכים במערכת יודעים בדיוק מתי מתחיל ונגמר כל סיבוב באמצעות שדר שמקבלים כל התהליכים באותו זמן. אנו מניחים שכל פעולה במערכת כזו לוקחת בדיוק סיבוב אחד, אין עיכובים.

**העברת הודעות מסונכרנת**: כל תהליך שולח הודעות בזמן . אנו מניחים שכל ההודעות שמתקבלות בתהליכי היעד נקראות ומעובדות עד סוף זמן , אמנם יכול להיות מצב שהודעות לא מגיעות.

**שיתוף זיכרון מסונכרן**: בכל round כל תהליך יכול לגשת לתא זיכרון אחד המשותף לכל התהליכים, ולכתוב או לקרוא אליו.

סיבוכיות הזמן לאלגוריתם במערכת מסונכרנת הוא מספר הסיבובים הדרוש כדי לסיים את האלגוריתם.

### מערכת לא מסונכרנת

זוהי מערכת שבה הזמן שלוקחת פעולה אינו ידוע אך כן ידוע שהזמן סופי. במערכת כזו יכולים להיות עיכובים. אנו צריכים לשקול את המקרה הגרוע ביותר, לכן אנו מניחים שהעיכובים נקבעים על ידי יריב שמחפש להכשיל את המערכת.

**העברת הודעות לא מסונכרנת**: כל תהליך שולח הודעות וידוע שהן אכן מתקבלות ביעדן, אך משך העיכוב הוא שרירותי בין הודעה להודעה ונקבע ע"י יריב. אמנם היריב אינו יכול לקבוע ש-B יקבל הודעה מ-A לפני ש-A שלח אותה, הסדר נשמר.

**שיתוף זיכרון לא מסונכרן**: כל התהליכים לבסוף מסיימים את פעולתם, אך משך הזמן הדרוש שרירותי ונקבע ע"י יריב.

כיצד נגדיר סיבוכיות זמן לאלגוריתם במערכת לא מסונכרנת? לא נרצה שהגדרת הסיבוכיות תהיה תלויה בזמן שבוחר היריב שכן זמן זה לא תלוי באלגוריתם. לכן את כל טווח הזמנים שהיריב יבחר לעכב, ננרמל לערכים בין 0 ל-1 ביחידות זמן מתאימות. בדרך כלל נתעניין במקרה הגרוע ביותר שבו זמן העיכוב מקסימלי, לכן בדרך כלל נניח שכל עיכוב הוא בדיוק 1.

## סוגי תקלות

1. Crash - אחד התהליכים קרס בשלב מסוים בביצוע. במערכת מסונכרנת יכול להיות שהקריסה הייתה באמצע ה-round ולכן חלק מהתהליכים קיבלו הודעה וחלק לא.
2. Byzantine - תהליך לא צפוי ועושה מה שבא לו. זה יכול להיות בגלל פריצת האקרים אליו. במצב כזה ניתן לעשות יותר נזק מאשר אם התהליך היה קורס. מערכת שיודעת להתמודד עם תקלה כזו תהיה עמידה בפני כל התקלות.
3. Omission - תהליך או קו תקשורת הפסיקו לעבוד אך רק באופן זמני. לדוגמה, חלק מההודעות הולכות לאיבוד.
4. Resilience - מספר תהליכים קרסו ולא רק אחד. במקרה זה המערכת צריכה להראות עמידות ולהמשיך לתפקד.

## תכונות נצרכות

כאשר מתמודדים עם מערכות ופרוטוקולים מבוזרים, ישנם מאפייני נכונות מסוגים שונים שצריכים להיות למערכת:

1. Safety - המערכת לעולם לא תבצע דברים רעים שאנו רוצים להימנע מהם. כדי להוכיח שהמערכת מקיימת תכונה זו יש להוכיח באמצעות אינווריאנטה שהמצב ההתחלתי בטוח, ולאחר מכן כל פעולה חוקית גם כן מביאה למצב בטוח.
2. Liveness - זוהי תכונה משלימה לבטיחות, שכן כדי לקיים בטיחות אפשר גם לא לעשות כלום. תכונה זו דורשת שמשהו טוב כן יקרה למישהו, כלומר המערכת תיתן לפחות לתהליך אחד לפעול. שקול ל-deadlock free.
3. Fairness - מוסיפה על Liveness שלא מספיק שמשהו טוב יקרה לתהליך אחד אלא לבסוף צריכה לעשות טוב לכולם, כלומר כל התהליכים צריכים לפעול. שקול ל-starvation free.

לדוגמה, עבור מערכת של רמזורים בצומת, המערכת תהיה בטוחה אם בכל זמן לכל היותר רמזור אחד מציג אור ירוק. כדי שהמערכת תקיים Liveness צריך שתמיד לפחות רמזור אחד יציג אור ירוק. וכדי שהמערכת תהיה הוגנת כל הרמזורים צריכים להיות מתישהו ירוקים.

## העברת הודעות - Massage Passing

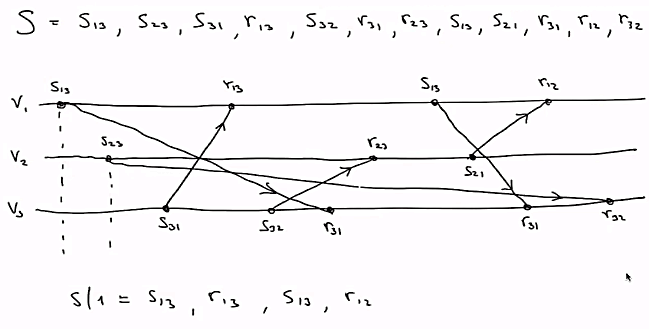
נגדיר את מודל העברת הודעות בצורה פורמלית יותר כדי שנוכל בהמשך לדון על נכונות מערכת כזו.

**הגדרת המודל**: במערכת העברת הודעות מכילה n צמתים , אנו מניחים שאנו יודעים את n. בכל זמן כל צומת נמצא באיזשהו מצב , התלוי בקלט לצומת, משתנים מקומיים שלו, שעון פנימי, והיסטוריית האירועים שהצומת הזו גילה. יש שלושה סוגים של אירועים:

* אירוע שליחה - של הודעה. האירוע גלוי אך ורק לצומת ששלח את ההודעה ולא למקבל (אצלו יהיה אירוע קבלה).
* אירוע קבלה - של הודעה. האירוע גלוי אך ורק לצומת שקיבל את ההודעה, השולח לא יודע שההודעה התקבלה.
* אירוע תזמון - זהו אירוע שהופעל בצומת על ידי שעון פנימי כלשהו.

### הגדרות נוספות

* Configuration: היא וקטור של כל המצבים של n הצמתים בזמן נתון - . מסומנת ב-C.
* Execution Fragment: קטע הרצה שמיוצג באמצעות רצף קונפיגורציות ואירועים לסירוגין - . כאשר לא חייבים להתחיל מ- שהיא קונפיגורציה התחלתית, וכל שלשה היא עקבית, כלומר כל אירוע הוא שמוביל מקונפיגורציה אל .
* Execution: זהו קטע הרצה שחייב להתחיל מ-.
* Schedule: זהו Execution רק בלי הקונפיגורציות, רק האירועים - . במילים אחרות, לוח הזמנים מכיל את האירועים של שליחה וקבלת הודעות עם הקלטים שלהם. במידה והרצף גם הגיוני, כלומר אין הודעה שנשלחת לפני שמתקבלת, נאמר שזהו Admissible Schedule, כלומר לוח זמנים קביל. כיוון שהמודל דטרמיניסטי מרצף האירועים ניתן לדעת גם את הקונפיגורציות, לכן השימוש ב- Scheduleיותר נוח. אנו תמיד מניחים שהאירועים נקבעו על ידי יריב ומסודרים לפי המקרה הגרוע ביותר.
* Schedule Restriction: בהינתן לוח זמנים S נגדיר את ההגבלה של צומת , המסומן ב-, להיות כל האירועים ב-S ש- יודע עליהן. נזכיר שאם צומת שלח הודעה לצומת , אזי במידה ו- יקבל את ההודעה לא ידע על כך. כמו כן כמובן שאינו יודע על אירועי שליחה של צמתים אחרים. דוגמה בשקף 26 במצגת 1.

צורה גרפית נוחה שמראה את ה-של כל הצמתים, היא לצייר לכל צומת i ציר זמן שעליו נכתוב את כל המאורעות שראה לפי הסדר. בין כל אירוע שליחה מ-i ל-j ואירוע קבלה מתאים , נמתח חץ כדי להראות את כיוון השליחה. באמצעות גרף זה אפשר לקבל את שכל צומת ראה בציר הזמן שלו, וכן להבין את סדר השליחה והקבלה בהרצה ולבדוק האם סדר זה אכן הגיוני.

### משפט אי ההבחנה (Indistinguishability)

בהינתן שני לוחות זמנים S ו-S’ של שתי הרצות שונות על אותה מערכת מבוזרת, ותהליך (צומת) כך שמתקיים: , כלומר רואה אותם אירועים וקלטים בשתי ההרצות, אזי יבצע בדיוק אותם פעולות ב-S וב-S’ שכן הוא אינו יכול להבחין בהבדל ביניהם. *במערכת רגילה משפט זה טריוויאלי. אמנם במערכת מבוזרת, שבה יכול להיות שכל שאר התהליכים מלבד בשני ההרצות עשו פעולות שונות, החידוש הוא ש- יעשה בשניהם בדיוק אותו דבר.*

***הגדרה:*** *עבור שתי הרצות* E *ו*-E’*, אם קיים צומת שבשניהם רואה את אותם* אירועים וקלטים, , אזי נאמר ששתי ההרצות דומות עבור , בסימון .