8. SYNCHRONOUS AGREEMENT

Magnus Jensen

- Motivation, clockdrift
- Round Based protocols
  - Tid
  - Runder
  - Protokol
- The Scheduled Broadcast Problem
  - Egenskaber
    - Validity
    - Agreement
    - Termination
- Dolev Strong løsning
  - Ideen
  - Protokollen
    - Egenskaber

SYNCHRONOUS AGREEMENT

1. Motivation, clockdrift

Da jeg startede her på universitet; gik jeg rundt med et Apple Watch - et ur der er præcist ned til 50 millisekunder.

Der gik nogle måneder; og så skiftede jeg mit Apple Watch ud med et Hamilton Khakifield, et automatisk mekanisk ur - og det mister op til 20 sekunder om dagen.

I min dag til dag tilværelse, betyder det ikke meget; men for en computer kan det betyde forskellen på liv eller død.

Specielt når vi tænker på rumskibe, fly, sundhedsvæsnet osv.

2. Round Based protocols

2.1 Tid

Hvis vi skal have **flere parter** i et **distribueret system** til at arbejde efter tid; som synchronous agreement handler om; så **kræver det** at de på sigt til dels er **enige om hvad klokken er**.

Så hvis vi har en **leder i systemet**, med en **præcis klok** - så lader vi da bare alle de **andre parter i systemet spørge** lederen om hvad klokken er! Yes det gør vi!

Åh nej vent... Det kommer jo ikke til at virke; for vi ved ikke hvor lang tid det **tager at spørge og svare** - så det svar vi får fra lederen; passer jo ikke mere. Og hvis vi er **mange der spørger**; og får svar på **vidt forskellige tidspunkter** - har vi en masse forskellige billeder af hvad klokken er!

Det viser sig at være yderst svært at blive enige om hvad klokken er.

2.2 Runder

Antager vi, at man kan blive enige om **hvad klokken er**; så kan man lave et distribueret system **der arbejder på runder**.

Ideen er:

> *I hver runde, kan hver part sende en besked*

Men for det kan virke, skal vi tillade følgende:

> - ***Offset**: Hvor meget et ur drifter*
> - ***Trans**: tiden det for at sende*

Man kan da **vide**, om en **besked er sendt** eller ej efter

> $t + 2Offset + Trans$

Da begge klokke kan drifte.

Derved kan vi ved disse **bounds, undgå at miste beskeder**.

2.3 Protokol

Hvis vi kender:

> - *MaxTrans*
> - *MaxOffset*
> - *MaxComputation*

Kan vi sætte timeouts for hvornår en besked der er sendt i en runde, skal være kommet.

En runde har derfor længden:

> $slotLength = 2MaxDrift + MaxTrans + MaxComputation$

Så starter en runde *r* ved:

> $start = T0 + r * slotLength$

Og slutter ved

> $slut = T0 + r * slotLength + slotLength$

Og runden løber da fra

> $Runde^i = [start^i; slut^i[$

En protokol[1] med runder forløber da således:

> - *Alle starter i runde 0*
> - *Start beskeder sendes ved* $slut^0$
> - *Runde* ***r*** *gemmer Pi første besked fra Pj fra runde* ***r-1***
> - *Udregn og send til alle*

Modtager vi ikke en besked mellem

> $[start - 2MaxDrift; start + MaxTrans + 2MaxDrift]$

Anser vi den part som crashet eller ond.

3. The Scheduled Broadcast Problem

Så nu når vi har runder på plads; så lad os **snakke om et problem**: The Scheduled Consensus Broadcast Problem.

> *Pj ved Pi skal sende en besked klokken t; hvorefter Pj skal levere den*

For en protokol til korrekt at udføre dette, skal alle ærlige parter køre protokollen i samme runde.

3.1 Egenskaber

3.1.1 Validity

> - ***Validity**: Hvis B er korrekt, så er alle ærlige parters resultat m*

3.1.2 Agreement

> - ***Agreement**: Alle ærlige parter vil udregne samme resultat*

3.1.3 Termination

> - ***Termination**: Alle ærlige parter vil på sigt udregne et resultatet*

4. Dolev Strong løsning

Dolev Strong protokollen kan gøre dette ved at **bruge signature**.

Den er til for authentikeret broadcast ved **t < n**.

> *Dolev Strong: $t < n$*

Den virker ved at bruge signature under et public-key system; hvorfor protokollen kun er så sikker som key systemet er.

4.1 Ideen

Ideen ved protokollen; er at ud fra hvad en broadcaster B siger, så vil alle udskrive beskeden hvis *m*:

> - *I runde **r** har **r** unikke signature*
> - *En signatur fra B*

4.2 Protokollen

Generelt:

> - *Bekseder sendes med en signatur*

Hver Pi holder nogle variabler:

> - *Resultat; kaldet Ri*
> - *Videsendte beskeder; kaldet Vi*

Indledenderunde (1):

> - *D flooder* `(m, {SIGNd(m)})` *og sætter* ***Rd = m***

I runderne: *2, ..., n + 1*:

> - *Pj modtager en* ***m*** *og et sæt af signature* ***S****, hvor* ***Vj*** *ikke indeholder* ***m****, så er* ***S*** *valid hvis:*
> - *1) Alle kan verificeres til* ***m***
> - *2) En er fra D*
> - *3) Antallet er* ***r*** *≤ unikke;*
> - *Og derfor så signere* ***m*** *og send* `(m, S + SIGNj(m))` *til alle; samt tilføj* ***m*** *til* ***Vi***

Udgangsrunde:

> - *For hver Pj med EN slags besked i Vi, så sæt Rj = m, ellers Rj = NoMsg*

4.2.1 Egenskaber

Det er nemt at se at vi har **termination**; da **alle sætter Rj til sidst**.

For at argumentere for **Agreement, så ser vi på runden r hvor korrekte Pi tilføjer m til Vi**. Sættet af signature **S er da validt; hvorfor S**' der sendes videre vil være **validt**.

Hvis r var sidste runde; vil Pj allerede **have m i Vj**, og hvis der er en **runde mere**, vil **Pj tilføje m til Vj** i den runde, **da S' er valid**.

For **Validity** gælder det; at en S kun kan være valid sålænge B signerede m. Så siden B kun signere m, så vil ingen ærlig P have `v != m`.

---

1. Mange detaljer er taget fra her