



Parallelle og distribuerte databaser – del II

- Paxos consensus
- Paxos commit
- Databaser i peer-to-peer-systemer (P2P)
- Databaser i mobile ad-hoc-nettverk (MANET)

Paxos consensus I

- Utgangspunkt: Et distribuert system av N samarbeidende noder som trenger å oppnå konsensus/komme frem til en felles beslutning.
 - Noder kan feile og meldinger bli borte
 - Samme melding kan bli levert flere ganger
 - En melding kan bruke vilkårlig lang tid på å bli levert
- **Paxos consensus** er en algoritme som består av en eller flere avstemningsrunder der målet er at nodene skal vedta ett felles **dekret**. (Dekretet representerer eller beskriver den felles beslutningen.) Hvis en majoritet av nodene blir enige om et dekret, blir det vedtatt.

Paxos consensus II

- I hver avstemningsrunde foreligger ett forslag til dekret. Dekretet kan variere fra runde til runde. Bare ett av dekretene som fremmes, kan bli vedtatt.
- I hver runde kan en node enten stemme for dekretet eller la være å stemme.
- I utgangspunktet er alle noder positivt innstilt til alle forslag til dekreter. Hvis en node ikke besvarer en melding, er det fordi ett av følgende har skjedd:
 - meldingen eller svarmeldingen gikk tapt
 - noden gikk ned før den fikk meldingen eller før den fikk svart på meldingen
 - noden velger å ikke besvare meldingen

Avstemningsrundene

- Hver avstemningsrunde tilordnes et rundenummer b . Første runde har $b = 0$.
- Flere runder kan pågå samtidig hvis en node velger å påbegynne en ny runde, typisk fordi den ikke fikk inn nok svar i foregående runde. Nye runder kan også påbegynnes av andre noder, f.eks. hvis de tror at den noden som startet algoritmen, har feilet.
- En node som påbegynner en ny runde, vil tilordne denne et rundenummer som er høyere enn de rundenumrene noden alt kjenner til.
- To noder p og q kan komme til å tildele samme rundenummer til to forskjellige runder. Da lar vi i såfall den runden som ble initiert av noden med lavest nodeid, være den med lavest rundenummer:
$$(b,p) < (b',q) \equiv b < b' \vee b = b' \wedge p < q$$
- Rundenummeret reflekterer ikke nødvendigvis i hvilken rekkefølge rundene faktisk gjennomføres.

Fasene i en avstemningsrunde

1. **Forberedelsesfasen:** En node ønsker å initiere en ny avstemningsrunde. Noden innhenter informasjon fra et tilstrekkelig stort antall noder om i hvilken runde de sist stemte, og hvilket dekret som da var oppe til avstemning.
2. **Stemmefasen:** Noden administrerer en avstemning der dekretet det skal stemmes over og hvilke noder som skal være med i avstemningen (et **kvorum** Q), delvis blir bestemt utifra de svarene som ble innhentet i fase 1.
3. **Vedtaksfasen:** Hvis alle i kvorumet stemte, er dekretet vedtatt, og noden sender dekretet til samtlige noder.

Variable og initialverdier

- $\text{lastTried}_p = b$
b er rundenummeret til siste runde initiert av node p.
Initielt: -1
- $\text{prevVote}_p = (b, q, d)$
b er høyeste rundenummer blant de rundene der node p har stemt. q er noden som initierte runde b. d er dekretet som var oppe til avstemning i runden. Initielt: (-1, nil, BLANK)
- $\text{nextBallot}_p = (b, q)$
b er høyeste rundenummer blant de rundene der node p har sagt seg villig til å delta. q er noden som initierte runde b.
Initielt: (-1, nil)
- $\text{outcome}_p = d$
d er det vedtatte dekretet. Initielt: BLANK

Fase 1 – forberedelsesfasen

1a. Node p velger en ny $b > \text{lastTried}_p$.

- $\text{lastTried}_p := b$.
- Send meldingen $\text{NextBallot}(b,p)$ til en mengde noder M der M utgjør en majoritet av nodene.

1b. Node q mottar en $\text{NextBallot}(b,p)$ -melding:

- Hvis $(b,p) > \text{nextBallot}_q$:
 - $\text{nextBallot}_q := (b,p)$.
 - Send meldingen $\text{LastVote}(b,p,\text{prevVote}_q)$ til p.
- Ellers: Ignorer meldingen.

Fase 2 - stemmefasen

2a. Node p har mottatt en $\text{LastVote}(b,p,v)$ -melding med $b = \text{lastTried}_p$ fra alle noder i en mengde $Q \subseteq M$ der Q utgjør en majoritet av nodene:

- Velg et dekret d (se neste side).
- Send meldingen $\text{BeginBallot}(b,p,d)$ til hver av nodene i Q.

2b. Node q mottar en $\text{BeginBallot}(b,p,d)$ -melding:

- Hvis $(b,p) = \text{nextBallot}_q$:
 - $\text{prevVote}_q := (b,p,d)$
 - Send meldingen $\text{Voted}(b,p,q)$ til p.
- Ellers: Ignorer meldingen.

Fase 2 – valg av dekret

- Hvis samtlige $\text{LastVote}(b,p,v)$ har $v = (-1, \text{nil}, \text{BLANK})$, kan dekretet i avstemningsrunde b velges fritt. Da vil node p fremme sitt eget forslag til dekret.
- Hvis det finnes minst én $\text{LastVote}(b,p,v)$ der $v = (b',q',d')$ har $b' \geq 0$ (da er $q' \neq \text{nil}$ og $d' \neq \text{BLANK}$), så finn den med størst b' og sett dekretet lik tilhørende d' .
- Dette, sammen med at kворумmet Q i hver runde må omfatte en majoritet av nodene, sikrer at avstemningsrundene vil konvergere mot et felles dekret.

Fase 3 - vedtaksfasen

- 3a.** Node p har mottatt en $\text{Voted}(b,p,q)$ -melding med $b = \text{lastTried}_p$ fra hver av nodene i Q:
 - $\text{outcome}_p := d$
 - Send meldingen $\text{Success}(d)$ til samtlige noder.
- 3b.** Node q mottar en $\text{Success}(d)$ -melding:
 - $\text{outcome}_q := d$

Initiering av algoritmen

- Hvis ingen node er spesielt utpekt til å starte protokollen, kan en hvilken som helst node starte protokollen i fase 1 med $b = 0$.
- Hvis én av nodene er utpekt til å starte protokollen, er fase 1 i første avstemningsrunde unødvendig fordi alle nodene vil besvare den med en $\text{LastVote}(b,p,v)$ der $v = (-1, \text{nil}, \text{BLANK})$.
 - Protokollen starter med at noden utfører trinn 2a med $b = 0$ og $d = \text{det dekretet noden ønsker å fremme forslag om}$.
 - Testen i trinn 2b endres til “Hvis $b = 0$ og $\text{nextBallot}_q = (-1, \text{nil})$ eller hvis $\text{nextBallot}_q = (b,p)$ ”.
 - Ytterligere avstemningsrunder starter med fase 1.

Logging

- Hver node må logge sine operasjoner og persistere loggen før meldinger sendes slik at tilstandsinformasjon (innholdet av de lokale variablene og hvilke meldinger som er sendt) ikke kan gå tapt.

Konsistens og vranglås

- Algoritmen sikrer konsistens, dvs. alle noder der $\text{outcome}_p = d \neq \text{BLANK}$, har samme verdi d .
- Algoritmen hindrer vranglås (deadlock):
Til ethvert tidspunkt er det alltid mulig å gjennomføre en runde som leder til et vedtak, såsant tilstrekkelig mange meldinger kommer frem og besvares i hver fase.
- Algoritmen er ikkeblokkerende¹ for $N \geq 3$ (der N er antall noder som deltar).

¹En protokoll er **blokkerende** hvis det at én strategisk node går ned på et kritisk tidspunkt, er tilstrekkelig til å få protokollen til å «henge».

Alternative kvora

- For å sikre konsistens er det tilstrekkelig at kvorumet Q i fase 2 velges slik at hvis vi ser på mengden av samtlige kvora (dvs. for alle gjennomføringer av fase 2, og uansett hvilken node som gjennomfører fasen), så har to vilkårlige kvora blant disse alltid minst én node felles.
- Dette er trivielt oppfylt hvis alle velger $Q =$ en majoritet av nodene (dvs. slik algoritmen er formulert på de foregående sidene).
- Det er også mulig å gi hver node en vekt og definere Q til å være en majoritet hvis summen av vektene til nodene i Q er større enn halvparten av totalvekten av nodene.

Fremdrift

- I utgangspunktet kan en node fritt velge å la være å besvare meldinger. Dette påvirker ikke konsistensen, men kan hindre at nodene oppnår konsensus/kommer frem til en beslutning.
- For å sikre at algoritmen har fremdrift, må alle noder forplikte seg til å gjennomføre trinnene 1b, 2a, 2b og 3a så raskt de kan (besvare meldinger og følge opp protokollen så lojalt som mulig).
- I tillegg må, så lenge det ikke er vedtatt noe dekret, nye runder (trinn 1a) initieres fra tid til annen.
- Nye runder må imidlertid ikke initieres for ofte: Hvis det stadig initieres nye runder, risikerer man at ingen av rundene noensinne lykkes (livelock).

Leder

- Det er enklest å møte utfordringene knyttet til nye avstemningsrunder hvis en av nodene velges til leder og bare lederen får initiere nye avstemningsrunder.
- I tillegg må man ha en protokoll for valg av ny leder hvis ledernoden går ned.
- Algoritmen er konsistent også hvis flere noder tror de er ny leder.
- Algoritmen er fortsatt ikkeblokkerende.

Bruk av Paxos consensus

- Avgjøre om en distribuert transaksjon kan committe – **Paxos commit**
- Distribuert låsehåndtering
- Avgjøre i hvilken rekkefølge oppdateringer av et replikat skal gjennomføres
- Synkronisering av klokker
- Koordinering av høynivåprotokoller
- Synkronisering av faser i distribuerte algoritmer

Paxos commit

- Anta i det videre gitt en distribuert transaksjon som involverer K noder. Transaksjonen kan committe bare hvis samtlige subtransaksjoner kan committe.
- Paxos commit-protokollen benytter følgende prosesser:
 - Én **ressurshåndterer** på hver node som deltar i transaksjonen.
Totalt: K ressurshåndterere på K forskjellige noder
 - En **leder**
 - N **akseptorer**.
Totalt: N akseptorer på N forskjellige noder

Hovedidé

- Lederen sender en **Prepare**-melding til alle ressurs-håndtererne.
- Ressurshåndtererne bestemmer seg for om de kan committe eller må abortere, og starter på vegne av lederen opp hver sin instans av Paxos consensus-protokollen med dekretet «**prepared**» eller «**aborted**».
- Lederen samarbeider med akseptorene om å gjennomføre alle Paxos consensus-instansene.
- Lederen formidler det samlede resultatet ved å sende en **Commit**- eller **Abort**-melding til alle ressurshåndtererne.
(Samtlige ressurshåndterere må ha startet opp sin instans av Paxos consensus med dekretet «**prepared**» for at den distribuerte transaksjonen skal kunne committe.)

Paxos consensus-instansene

- Når en ressurshåndterer kjenner utfallet av sin del av transaksjonen, bekjentgjør den dette ved å bruke **en egen, dedikert instans** av Paxos consensus-algoritmen med dekretet «**prepared**» eller «**aborted**».
- Formålet med instansen er å gjøre utfallet på denne spesifikke noden kjent i det distribuerte systemet, og på en måte som sikrer at protokollen er ikkeblokkerende.
- Med K noder blir det K samtidige instanser av Paxos consensus, én for hver node som er involvert i transaksjonen. For hver instans foretas det separate, uavhengige avstemningsrunder for å få gjort utfallet på noden kjent.

Akseptorene

- Akseptorene gjør jobben til de N samarbeidende nodene i Paxos consensus-algoritmen for *samtlige* K instanser av consensus-algoritmen.
- Antall akseptorer (N) velges etter hvor robust protokollen skal være: Med $N = 2F+1$ akseptorer tåler protokollen at opptil F av akseptorene går ned uten at protokollen blir hengende.

Initiering av protokollen

- Protokollen starter når den første ressurshåndtereren er klar til å committe sin del av transaksjonen.
 - Ressurshåndtereren sender en **BeginCommit**-melding til lederen, som sender en **Prepare**-melding til alle de andre ressurshåndtererne.
 - Ressurshåndtereren initierer deretter sin instans av Paxos consensus-algoritmen. (For detaljer, se neste side.)

Initiering av hver consensus-instans

- For å redusere antall meldinger og gjøre protokollen mer robust, er ansvaret for å initiere Paxos consensus-instansene delegert fra lederen til ressurshåndtererne:
 - Når en ressurshåndterer mottar en Prepare-melding, initierer den sin instans av Paxos consensus-algoritmen ved å sende en fase 2a-melding med rundenummer 0 og dekretet «**prepared**» eller «**aborted**» direkte til alle akseptorene.
 - Akseptorene responderer med fase 2b-meldinger til lederen.

Fremdrift

- Hvis det for en instans går for lang tid uten at lederen har mottatt fase 2b-meldinger fra en majoritet av akseptorene, initierer lederen fase 1a i en ny runde.
- Når tilstrekkelig mange akseptorer så har besvart med en 1b-melding, sender lederen fase 2a-meldinger til en majoritet av akseptorene.
- Hvis det fra fase 1b-svarene ikke forelå noe forslag til dekret, forslår lederen dekretet «**aborted**» i fase 2a.

Optimaliseringer

- For å redusere antall meldinger, kan hver av akseptorene vente til den har mottatt fase 2a-meldinger for samtlige Paxos consensus-instanser. Deretter sender akseptoren én akkumulert fase 2b-melding til lederen.
- Hvis en ressurshåndterer vet at dens subtransaksjon må aborteres, kan den kortslutte protokollen ved å sende **Abort** direkte til alle.

Prosessenes arbeidsoppgaver

- **Ressurshåndtererne** utfører subtransaksjoner på sine noder og avgjør om en subtransaksjon kan committe eller må aborteres.
- **Lederen** identifiserer det endelige (samlede) resultatet og distribuerer det til alle ressurshåndtererne. Lederen er ansvarlig for å drive protokollen fremover ved å initiere nye avstemningsrunder ved behov. Den **initielle lederen** plasseres gjerne på startnoden (den noden som initierte transaksjonen).
- **Akseptorene** deltar i avstemningsrundene og har ansvar for lojalt å følge opp fase 1a- og fase 2a-meldinger med henholdsvis fase 1b- og fase 2b-svar til lederen. Akseptorene er dessuten ansvarlige for å drive protokollen fremover ved å velge ny leder ved behov. Ny leder velges blant de nodene som har en akseptorprosess.

Om Paxos commit

- Paxos commit med én akseptor ($N = 1$) er ganske lik tofasecommit (2PC) når nettverk og noder fungerer feilfritt.
- Paxos commit er ikkeblokkerende for $N \geq 3$ - den tåler at en eller flere akseptornoder går ned hvis en majoritet av akseptornodene fortsatt er oppe. (I tillegg tåler den at en eller flere av de øvrige nodene går ned.)
- Ved valg av ny leder kan flere noder tro at de er den nye lederen. I ekstremtilfellet (avhengig av hvilken prosedyre som benyttes for ledervalg) kan hver av akseptornodene starte opp hver sin lederprosess. At flere noder tror de er leder, kan påvirke fremdriften til protokollen, men ikke dens konsistens: Protokollen kan bare konkludere med commit hvis samtlige ressurshåndterere har svart «**prepared**».

Peer-to-peer-systemer

- Et P2P-nettverk er et (virtuelt) nettverk av maskiner som
 - er **autonome**: noder kan bli med i eller forlate nettverket etter eget forgodtbefinnende
 - er **løst koblet**: kommunikasjon via et universalnett (f.eks. Internett)
 - har **lik funksjonalitet**: ingen leder- eller kontrollnoder
 - er villige til å **dele ressurser** med hverandre

Databaser i peer-to-peer-systemer

- En database i et P2P-nettverk må ivaretas kollektivt av nodene i nettverket. Krav til databasesystemet:
 - **Distribuert**: data er fordelt på nodene i nettverket
 - **Desentralisert**: alle nodene har samme administrative ansvar for databasen
 - **Feiltolerant**: systemet må være pålitelig selv om noder feiler og selv om den mengden av noder som utgjør nettverket, kontinuerlig endres
 - **Skalerbart**: systemet må være effektivt også for store nettverk

Databaser i P2P-systemer

- **Distributed hash tables** (DHT) er en klasse av teknikker for å realisere **oppslagstjenester i P2P-nettverk**
- Anta at databasen består av skjemaet $R(\underline{K}, V)$ der K er primærnøkkel og V tilhørende verdi. En DHT består av
 - en **nøkkelpartisjonering**, dvs. en hashfunksjon h
 - For hvert tuppel (k, v) i R brukes $h(k)$ til å beregne identiteten til den noden som har ansvaret for å lagre tuppelet (k, v)
 - et **overleggsnettverk (overlay network)**, dvs. en logisk struktur som knytter sammen nodene i nettverket
 - Hver node trenger bare å ha direkte kjennskap til en liten del av overleggsnettverket
 - Likevel kan noden som lagrer (k, v) , nås fra en hvilken som helst annen node med et lite antall meldingsutvekslinger

Eksempel på en DHT: Chord

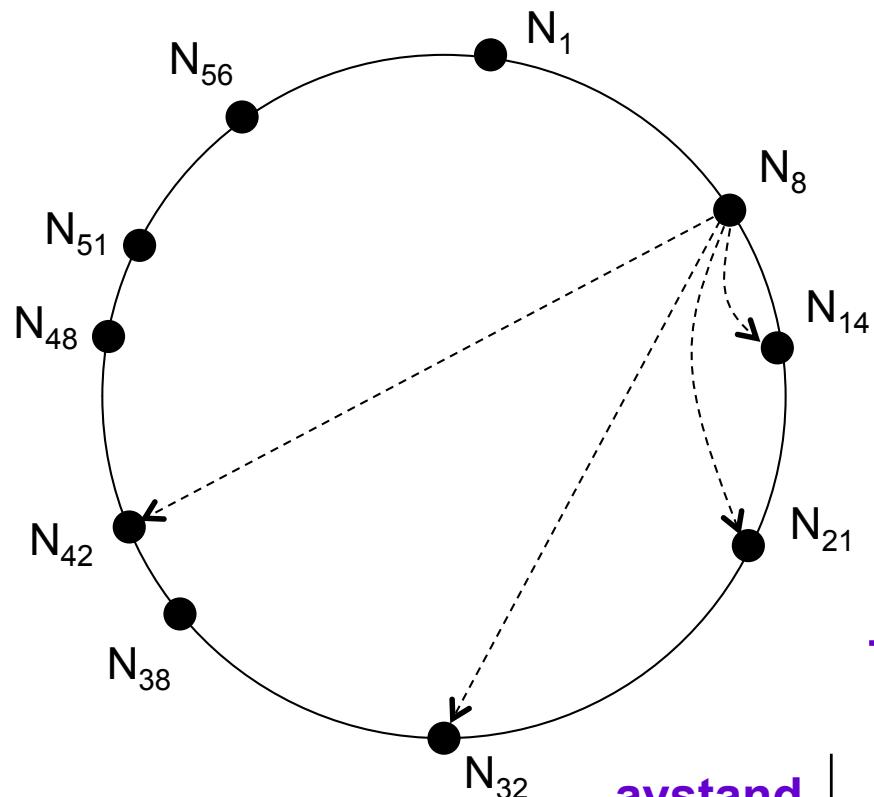
- Overleggsnettverket består av
 - en (logisk) **sirkel** som omfatter alle nodene i nettverket
 - strenger (**chords**) ”på tvers” i sirkelen
- Hashfunksjon:
 - $h(x) = x \text{ mod } 2^m$ for et passende heltall m
 - maksimalt antall noder er 2^m

Overleggsnettverket i Chord

- En node med ID w plasseres i sirkelen i posisjon $h(w)$
 - Vi antar at alle noder har en ID og at disse er entydige
- Hver node har en **fingertabell** med peker til de nodene som har posisjon $j+1, j+2, j+4, \dots, j+2^{m-1}$ i sirkelen, der j er nodens egen posisjon i sirkelen
 - Hvis det ikke er noen node i posisjon $j+2^i$ for en i , inneholder fingertabellen den noden med urviseren som er nærmest denne posisjonen. (Posisjonene beregnes modulo 2^m .)
 - Fingertabellen har lengde m og tar derfor ikke stor plass selv for store nettverk (dvs. stor 2^m).
- I tillegg har hver node en peker til sin forgjenger

Eksempel: $m=6$

Maksimalt $2^6 = 64$ noder



De stiplede pilene viser nodene som er med i fingertabellen til N₈. I tillegg vet N₈ hvilken node som kommer foran den i sirkelen.

Under er alle disse opplysningene samlet i én tabell.

Tabell for N₈:

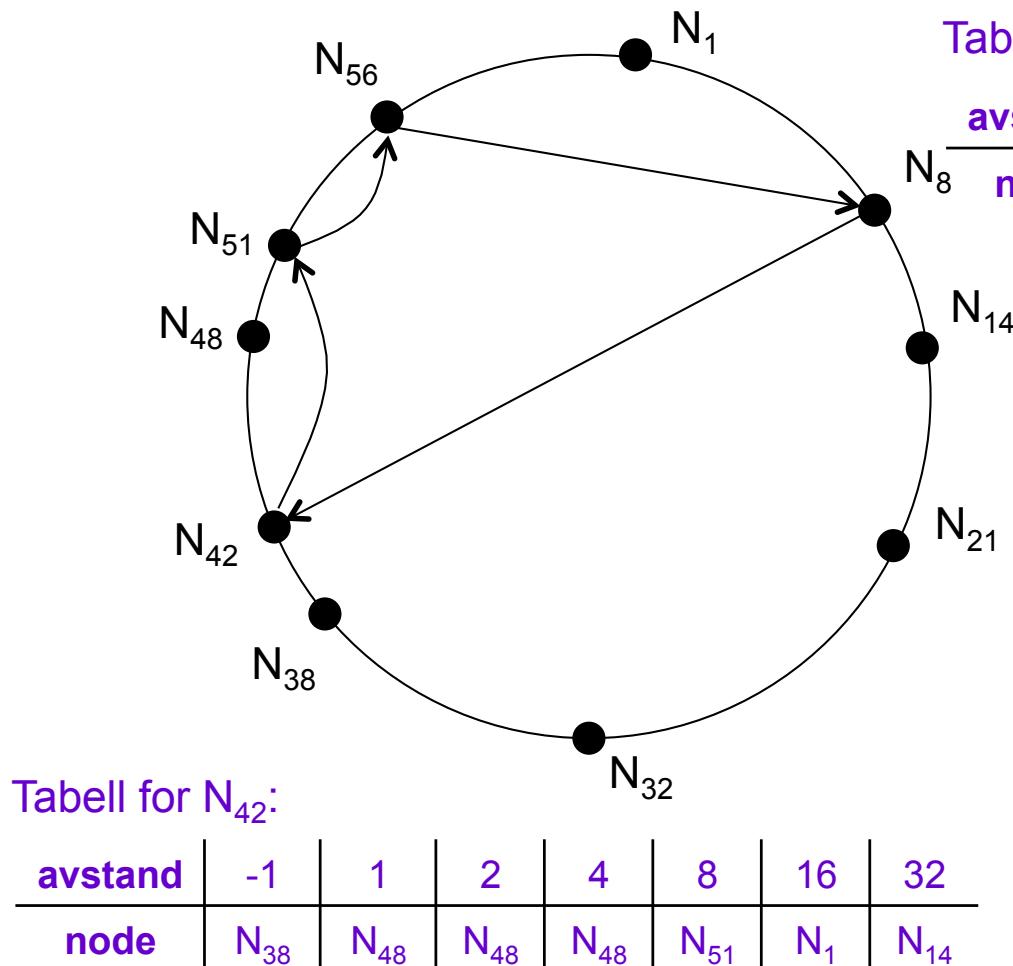
avstand	-1	1	2	4	8	16	32
node	N ₁	N ₁₄	N ₁₄	N ₁₄	N ₂₁	N ₃₂	N ₄₂

Datahåndtering i Chord

- Et tuppel (k, v) i tabellen $R(K, V)$ lagres på noden som har posisjon $h(k)$.
 - Hvis det ikke er noen node i posisjon $h(k)$, lagres tuppelet på den noden som er nærmest denne posisjonen (med urviseren).
- Hvordan finne et tuppel i nettverket: Anta at node N_i ønsker å finne v for en gitt k , men ikke har tuppelet lagret lokalt.
 1. Beregn $j = h(k)$. Sett $N_c = N_i$.
 2. N_c slår opp i tabellen sin og ser om den inneholder en node N_s der s er mindre eller lik j .
 - Hvis det ikke finnes noen slik s , er tuppelet (k, v) hos etterfølgernoden (N_e). N_c sender en melding til N_e og ber den om å returnere tuppelet direkte til N_i .
 - Hvis det finnes en slik s , så velg s størst mulig (men mindre eller lik j).
 - Hvis $s = j$, er tuppelet (k, v) hos N_s . N_c sender en melding til N_s og ber den om å returnere tuppelet direkte til N_i .
 - Hvis $s < j$, må tuppelet (k, v) befinne seg i en posisjon lenger ut i sirkelen enn s . N_c sender en melding til N_s og ber den om å overta rollen som N_c . Gjenta punkt 2.

Det trengs maksimalt $m+1$ meldinger: m forespørselsmeldinger + 1 melding med resultatet.

Eksempel: N₈ ønsker å finne (k,v) der h(k)=52



Tuppelet (k,v) ligger på den noden som ligger nærmest posisjon 52 (regnet med urviseren), dvs. node N₅₆. N₈ kan regne ut h(k)=52, men vet ikke at det ikke finnes noen node N₅₂, og heller ikke at tuppelet befinner seg i node N₅₆.

1. Nærmeste foran posisjon 52 i tabellen til N₈ er N₄₂. N₈ sender en forespørsel til N₄₂.
2. Nærmeste foran posisjon 52 i tabellen til N₄₂ er N₅₁. N₄₂ sender en forespørsel til N₅₁.
3. Siden etterfølgeren til N₅₁ er N₅₆, må tuppelet befinner seg der. N₅₁ ber N₅₆ sende tuppelet til N₈.

Totalt: 4 meldinger.

Mobile ad-hoc-nettverk

- Mobile ad-hoc-nettverk (**MANET**) er (lokale) trådløse nettverk som settes opp mellom noder som kommer i kommunikasjonsavstand til hverandre og som er villige til å kommunisere
- MANETs er karakterisert ved
 - hyppige endringer i den underliggende topologien
 - noder forflytter seg
 - naboskap endres
 - nettverk partisjoneres og smelter sammen
 - kommunikasjon skjer via radiobåndet
 - stor feilrate (kollisjoner av meldinger i eteren)
 - lav båndbredde
 - hver nettverkspartisjon består av i høyden 50-100 noder
 - mer enn dette er ikke praktisk håndterbart (pga. feilraten)

Databaser i mobile ad-hoc-nettverk

Krav til databasesystemet er omrent som for P2P-nettverk.
I tillegg er det ytterligere utfordringer knyttet til begrensninger i
nettverkskapasitet og hyppige endringer i nettverkstopologien:

- **Distribuert:** Data er fordelt mellom nodene. Optimalt antall og optimal plassering av replikater avhenger av nettverkstopologien og applikasjonsområdet.
- **Desentralisert:** Nodene må kollektivt ivareta det administrative ansvaret for databasen. Ved nettverkspartisjoner må databasen overleve i hver av partisjonene. Ved sammensmelting av nettverk må databasepartisjonene samordnes.
- **Feiltolerant:** Systemet må være pålitelig selv om noder feiler, naboskap endres, nettverket partisjoneres eller nettverk smelter sammen.
- **Skalarbart:** Størrelsen på hver nettverkspartisjon er begrenset, derfor er ikke dette kravet like fremtredende som i P2P-nettverk.
- **Konsistens:** MANETs er kjennetegnet av hyppige nettverkspartisjoner. Avveiinger mellom CP og AP i CAP-teoremet blir desto mer fremtredende.