Distribuerte databaser - kort sammendrag.

(Connly & Begg ch. 24).
(24.2 – er generelt om nettverk, bl.a. TCP/IP og forutsettes kjent, de andre protokollene er lite i bruk).

Hva?

Distribuert database:

Logisk relaterte data som er fysisk delt på flere steder i et nettverk.

NB! en sentralisert flerbrukerdatabase er ikke en distribuert database - men kan ha distribuert prosessering.

Distribuert databasesystem:

Databasesystem som håndterer en distribuert database og gjør distribusjonen transparent for brukerne.

Parallellprosessert databasesystem.

Databasesystem med flere prosessorer og disker i parallell, slik at de utnytter parallellprosessering.

Ofte brukt prinsipp ved distribuerte databaser:

Data skal lagres og behandles "nær" der de brukes.

Distribuerte databaser: fordeler og ulemper

Fordeler:

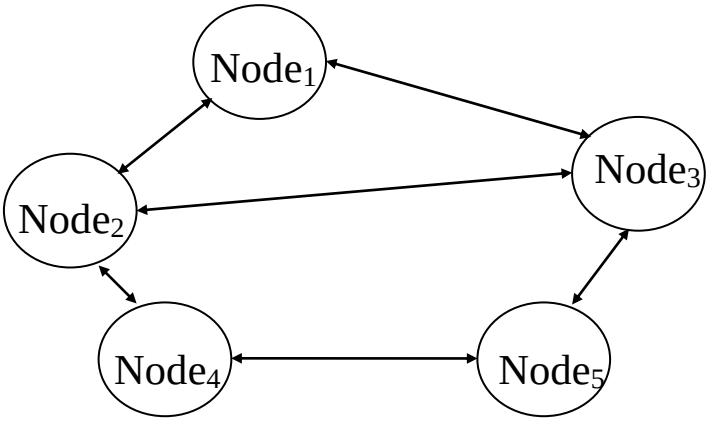
- Mer lik organisasjonsstrukturen
- Mer tilgjengelig - i alle fall de lokale data
- Raskere i mange tilfelle
- Kan gro distribuert
- Hvis full replikering: automatisk sikkerhetskopi for hverandre.

Ulempler:

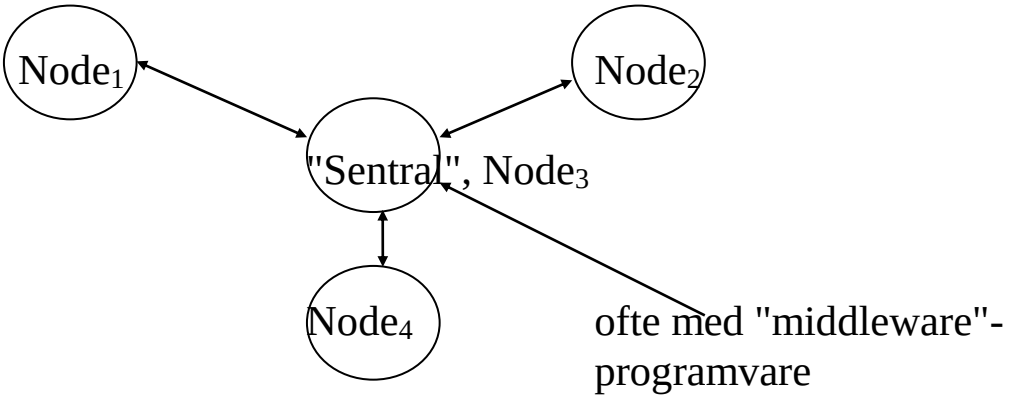
- Større kompleksitet
- Integritetssjekkene blir vanskeligere
- Skjemaevolusjon (endring av struktur) blir mer komplisert
- Tregere i noen tilfelle, bl.a. fører en join på distribuerte tabeller til mye nettverkstrafikk.
- Manglende standarder
- Lite erfaringsgrunnlag
- Gjerne dyrere

Ulik topologi:

Rent nettverk:



Stjernekobling:



eller **kombinasjon**.

NB.

- Kan være LAN og/eller WAN
- Hver node kan igjen være et fullstendig nettverk (cluster)
- I hver node kan det være data og/eller databasesystem.

Prinsipper for fragmentering:

Horisontal fragmentering = Restriksjon, σ

f.eks. slik at data plasseres fysisk i noden med

- sist besøkte avdelingsnr (enkelt - meget bra v/ stor stabilitet)
- mest brukt " " (statistikk?)
- postnr som antas å være nærmest aktuelt avdelingsnr.

Vertikal fragmentering = Projeksjon, π

Sted 1

Ansnr		

Sted 2

Ansnr	

f.eks. at ulike typer data om ansatte er distribuert.

Blanding av disse.

Eventuelt lagring av redundans overalt eller i utvalgte noder.

(NB: f.eks.triggere må brukes for å holde konsistensen)

I praksis: bør tenke på ytelse. Fordel hvis tunge koblinger mest mulig kan skje lokalt, f.eks.

- tabeller med mye brukt tung kobling ligger på samme sted
- koblingen kan distribueres (kun ved horisontal fragmentering)

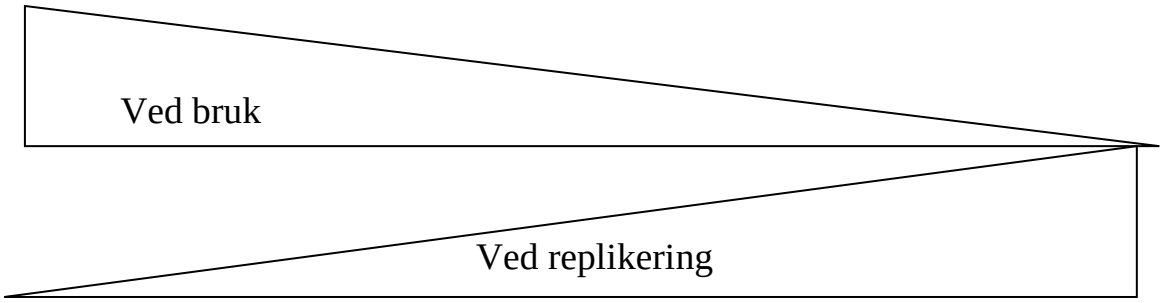
NB! Optimalisering kan være enda mer kritisk enn i sentraliserte databaser.

Grad av fragmentering og replikering.

Ingen dobbeltlagring	Endel dobbeltlagring	**Alt lagres overalt.**
Fullt fragmentert (evt. fullt sentralisert)	Kombinasjon av fragmentering, replikering, sentralisering	Fullt replikert

Hastighet ved søk/henting av data

Kommunikasjonskostnader/tid



Viktige kriterier for valg av konfigurasjon

- nødvendigheten av å ha 100% oppdaterte data
  - øyeblikkelig
  - på kort tid
  - f.eks. etter et døgn
- stabilitet av data
- bruk av data (f.eks. registrering, enkeltavgjørelser, statistikk)
- bruk av tellere (f.eks. ved nyregistrering)
- Koblinger og andre «tunge» operasjoner bør ikke skje distribuert.

Eksempler på konfigurasjoner

- tabeller med helt stabile data (kodetabeller etc.) legges lokalt de andre hentes fra sentralt sted - dvs. svært liten grad av distribusjon.
- data overføres fra et sentralt sted til de andre nodene i løpet av natten, lesing av de fleste data kan skje lokalt (data er "nesten" oppdaterte til enhver tid).
  Endring skjer til sentral database.
  Eks: Kan ligge en markør sentralt på når data sist var endret på personen. Hvis ikke endret etter replikering: les alt lokalt.
- sentralisering i LAN, fragmentering og/eller replikering i WAN.
- flerbruk med full fragmentering eller replikering
- sentralbord eller sentralbordløst

Ønsker for en distribuert/replikert database:

En distribuert database bør være transparent i forhold til:

Hva	Kommentar
• hvordan data er fragmentert	
• hvor data ligger	
• evt. replikering	
• transaksjoner	skal være like enten databasen er distribuert eller ikke.
• ytelse	neppe oppnåelig i praksis. Noe blir raskere, noe tregere.
• hvilken maskinvare som brukes i de ulike nodene	og hvilke nettverksprotokoller
• hvilke(t) databasesystem som brukes	inkl. om forskjellige system i de ulike nodene
• serialisering	
• låsing	inkl. oppdagelse av vranglås
• gjenoppretting (recovery)	

Det meste av dette er vanskeligere å håndtere i et distribuert miljø.

Kort sagt er ønsket:

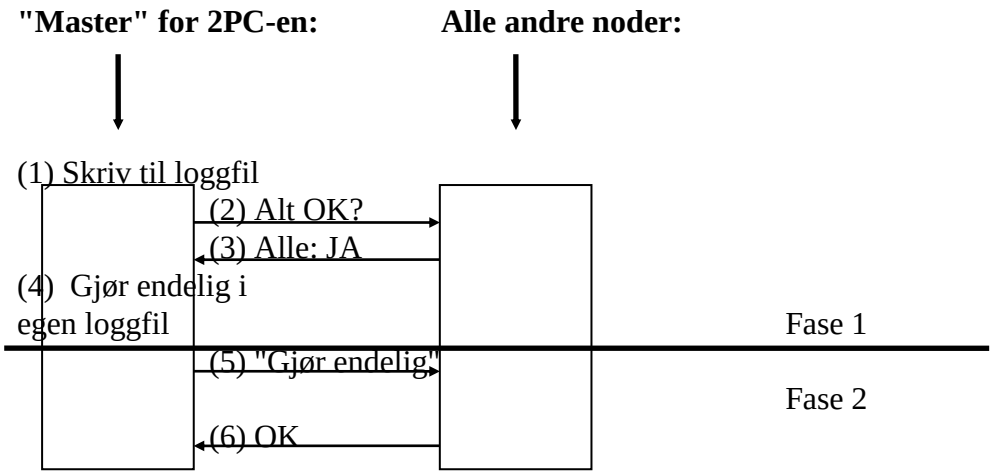
Databasemiljøet skal oppføre seg som det ikke var distribuert…

2-fase-gjennomføring (2PC) av distribuerte transaksjoner - enkel form.

Prinsipp: En transaksjon skal enten være helt gjennomført eller helt ugjort på alle steder samtidig.

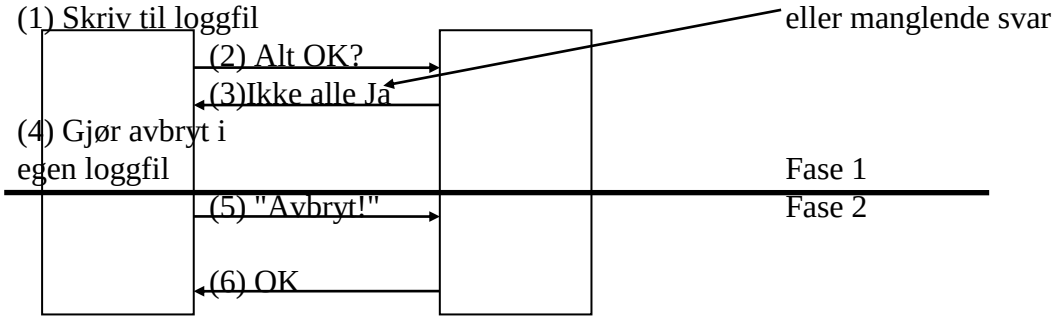
Etter at transaksjonen er skrevet (forhåpendligvis) i alle aktuelle noder, skjer gjennomføring / commit i 2 faser.

a) Gjennomføring ved alt ok:



Resultat: Kommitert endring på alle steder.

b) Gjennomføring ved feil en eller flere steder:



Resultat: Ingen endring på noen steder.

2PC, replikering og replikeringstjener.

2PC vil rulle tilbake transaksjonen dersom en node er nede, dvs. én fjerntliggende node kan ødelegge for alle de andre.

==>
Tendens til å gå over fra 2PC til replikering på de steder som er tilkoblet for øyeblikket, de andre replikeres så fort de kommer opp.
evt:
Kan bruke 3-fase-gjennomføring, 3PC-protokollen, inneholder en "pre-commit" i tillegg. Krever ikke at alle noder er oppe.

Endel systemer har sin egen **replication server** for å sørge for replikeringen.

Skyen, replikering m.m.

Egentlig ikke noen ny idé, web og epost har vært "i skyen" hele tiden.

Epost:

- **POP3:** epost ble hentet fra skyen til local maskin, behold kopi i skyen hvis du ønsker (➔ data kan hentes flere ganger). Statisk, ingen aktiv replikering.
- **IMAP:** hentes fra skyen, sammenlignes med lokal kopi og samordnes. ➔ kan ha lokale kopier, samtidig som de up-to-date. Dermed: standard replikering.
  NB! Kladd replikeres ikke (i alle fall ikke, i de systemene jeg kjenner til).

Dropboks som eksempel.

- Replikering:
  - PC/Mac/Linux: Normalt vilkårlig mange kopier, med full replikering. Kan utelukke enkeltmapper fra replikering.
  - Telefoner/pædder: Kun sentral lagring.
  - Rask, men ikke umiddelbar replikasjon.
  - Master er alltid i skyen.
- "Optimistisk låsning", risikerer "…'s conflicted copy ….", flere kopier, hvor ikke alle er like.

Andre skylagringssystemer – likt eller annerledes?

Spill over nettet

- Master på nettet, krevende m.h.t. datakommunikasjon

Kalendere/oppgavelister m.m.:

- Låsningstype varierer.

Avslutning: C.J. Date's regler for distribuerte databasesystemer.

RULE 1-LOCAL AUTONOMY

Definition: The sites in a distributed system should be autonomous or independent of each other .

Comments: A DBMS at each site in a distributed system should provide its own security, locking, logging, integrity, and recovery. Local operations use and affect only local resources and do not depend on other sites.

RULE 2-NO RELIANCE ON CENTRAL SITE

Definition: A distributed database system should not rely on a central site, because a single central site may become a single point of failure, affecting the entire system. Also, a central site may become a bottleneck affecting the distributed system's performance and throughput.

Comments: Each site of a distributed database system provides its own security, locking, logging, integrity, and recovery, and han- dles its own data dictionary. No central site must be involved in every distributed transaction.

RULE 3-CONTINUOUS OPERATION

Definition: A distributed database system should never require downtime.

Comments: A distributed database system should provide on-line backup and recovery, and a full and incremental archiving facility. The backup and recovery should be fast enough to be performed on- line without noticeable detrimental affect on the entire system per- formance.

RULE 4-LOCATION TRANSPARENCY AND LOCATION INDEPENDENCE

Definition: U sers and/or applications should not know, or even be aware of, where the data is physically stored; instead, users and/or applications should behave as if all data was stored locally.

Comments: Location transparency can be supported by extended synonyms and extensive use of the data dictionary. Location inde- pendence allows applications to be ported easily from one site in a distributed database system to another without modifications.

RULE 5-FRAGMENTATION INDEPENDENCE

Definition: Relational tables in a distributed database system can be divided into fragments and stored at different sites transparent to the users and applications.

Comments: Similar to the location transparency rule, users and applications should not be aware of the fact that some data may be stored in a fragment of a table at a site different from the site where the table itself is stored.

RULE 6-REPLICATION INDEPENDENCE

Definition: Data can be transparently replicated on multiple com- puter systems across a network.

Comments: Similar to the data location and fragmentation inde- pendence rules, replication independence is designed to free users of the concerns of where the data is stored. In the case of replication, users and applications should not be aware that replicas of the data are maintained and synchronized automatically by the distributed database management system.

RULE 7-DISTRIBUTED QUERY PROCESSING

Definition: The performance of a given query should be indepen- dent of the site at which the query is submitted.

Comments: Since a relational database management system pro- vides nonnavigational access to data (via SQL), such a system should support an optimizer that can select not only the best access path within a given node, but also can optimize a distributed query per- formance in regard to the data location, CPU and I/O utilization, and network traffic throughput.

RULE 8-DISTRIBUTED TRANSACTION MANAGEMENT

Definition: A distributed system should be able to support atomic transactions.

Comments: Transaction properties of atomicity, consistency, durability, isolation, and serialization should be supported not only for local transactions, but also for distributed transactions that can span multiple systems. An example of a distributed transaction management issue is transaction coordination in the distributed two-phase commit processing.

RULE9-HARDWARE INDEPENDENCE

Definition: A distributed database system should be able to oper- ate and access data spread across a wide variety of hardware platforms.

Comments: Any truly distributed DBMS system should not rely on a particular hardware feature, nor should it be limited to a cer- tain hardware architecture or vendor.

RULE 10-OPERATING SYSTEM INDEPENDENCE

Definition: A distributed database system should be able to run on different operating systems.

Comments: Similar to Rule 9, a truly distributed database system should support distribution of functions and data across different operating systems, including any combination of such operating sys- tems as DOS, UNIX, Windows NT, MVSNM, VSE, and V AX.

RULE 11-NETWORK INDEPENDENCE

Definition: A distributed database system should be designed to run regardless of the communication protocols and network topology used to interconnect various system nodes.

Comments: Similar to Rules 9 and 10, a truly distributed database system should support distribution of functions and data across dif- ferent operating systems irrespective of the particular communica- tion method used to interconnect all participating systems, including local and wide area networks. In fact, networks and communication protocols can be mixed to satisfy certain business, economic, geo- graphical, and other requirements.

RULE 12-DBMS INDEPENDENCE

Definition: An ideal distributed database management system must be able to support interoperability between DBMS systems running on different nodes, even if these DBMS systems are unlike (heterogeneous).

Comments: All participants in distributed database management systems should use common standard interfaces (APIs) in order to interoperate with each other and to participate in distributed data- base processing.