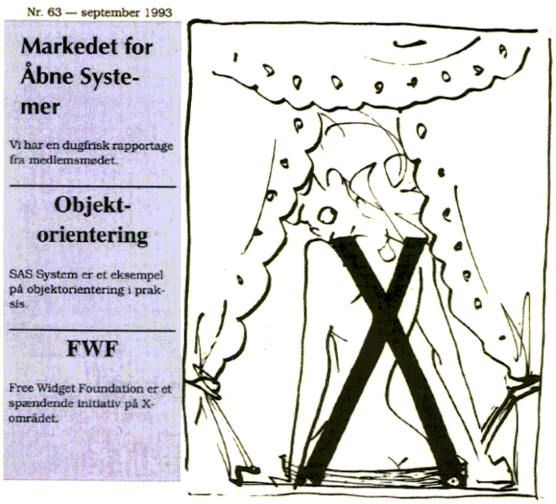
DKuug nr. 172 *Oktober 2013*

NYT

DKUUG-Nyt


Nr. 63 — september 1993

Markedet for Åbne Systemer

Vi har en dugfrisk rapportage fra medlemsmødet.

Objekt-orientering

SAS System er et eksempel på objektorientering i praksis.

FWF

Free Widget Foundation er et spændende initiativ på X-området.

For 20 år siden var X11 tillokkende - er hun det stadig?

```
void loop()
{
  if (Serial.available() > 0)
  {
    inByte = Serial.read();
    if (inByte == 115)
    {
      //Return status on door
      buttonState = digitalRead(3);
      if (buttonState == LOW)
        Serial.print("CLOSED\n");
      else
        Serial.print("OPEN\n");
    }
    else if (inByte == 108)
    {
      if (ledOn)
      {
        digitalWrite(13,LOW);
        ledOn = false;
...
```

Dansk Forum for Åbne Systemer

DKUUG - Unix Brugere (Linux/BSD) system administratorer

Community for IT-specialister og IT-interesserede.

DKUUG

Pebersvenden ser tilbage

30 år og stadig ungkarl

Unix er ca. 40 år gammel hvis man tæller tilløbene med. Den Unix, som de første pionerer i DKUUG kendte det, kom som bånd fra Berkeley (BSD, Berkeley Software Division)i ca. 1977-78-79 for akademiske formål, mens Unix System V blev kommercielt released fra AT&T (American Telephone & Telegraph) i 1983. Altså 30 år.

De gamle Unix-varianter er for længst ude af billedet, men det er principperne, som tæller, og de lever i bedste velgående:

- Små programmer, som kan én ting, er bedre end store programmer, som kan en hel masse.
- Mange små programmer kan opfylde uforudsete behov, hvis de kan arbejde sammen via et kommandoliniesprog.
- Modularisering af systemet på flere niveauer sikrer levedygtighed, idet et modul kan skiftes ud med et andet, når behov opstår eller fejl skal rettes.
- Dokumentationen afspejler i få ord de vigtigste ting.

DKUUG blev stiftet 18.november 1983. Der gik nogen tid inden foreningen i 1984 fik sit første blad - se artiklen side 4.

Selv om der er gået 30 år er DKUUG ikke helt overflødig. Unix ideen er spillevende i server og desktopmiljøer, på routere, embedded systemer og smartphones. Linux og BSD vedligeholdes og videreudvikles af en større skare kompetente programmører.

Det er velkendt at brugerne trækker i én retning og leverandørerne i en anden; forståelse af udtrykket "teknisk lock-in" er ved at brede sig. Unix i form af Linux, Gnu og BSD er et modtræk mod lock-in.

Derfor er det så vigtigt at brugerne er repræsenteret i en eller flere sammenhæng, ikke at DKUUG ene og alene kan opfylde det behov, men vi prøver at gøre os nyttige og hjælper så godt vi kan.

Hvorfor er Unix stadig moderne?

Først og fremmest: Dokumentationen er ærlig! Der er ikke andre systemer, som lige så præcist giver, hvad man har brug for: En one-liner, en synopsis, og en 4-liniers beskrivelse; options, henvisning til relevante programmer, libraryfunktioner og/eller filformater, og advarsel mod svage punkter - bugs! Men der er mere: Systeminformation deles op i 8 eller 9 sektioner, se man(1), og der er desuden reference - dokumentation og bruger-introduktion (user-guide).

Noget fandt man også hos andre leverandører. Men manual-sidernes **bug**-afsnit fandt man imidlertid ***ikke*** andre steder!

Internet og DKuug

DKuug drev den første service som gav private og virksomheder adgang til Internet. Modemopkoblinger var måden, man gjorde det på for de fleste, TDC kunne eller ville ikke levere ADSL førend engang i slutningen af 90'erne, og det var endda først efter at man havde opgivet 64kbit med ISDN. Jeg har så sent som 1999 haft adgang til telefon opkobling på DKuug's systemer. DK-net og DK-hostmaster blev købt af TeleDanmark eller TDC på et tidspunkt, hvor jeg ikke var medlem og ikke var blevet trukket ind i strømmen af begivenheder. Det samme skete i Holland - men til en anden og højere pris. DKuug har gennem de seneste år vist meget mere ansvar i forvaltningen af formuen, som nu sker som for et umyndigt barn gennem en autoriseret formueforvalter.

Standardisering og DKuug

DKuug har også arbejdet for danske specialtegn (især '**æ**', '**ø**' og '**å**') i det internationale tegnsæt.

For små 7 år siden skubbede vi på i forbindelse med standardisering af dokumentformater.

En af de erfaringer, vi gjorde os, var at der er uhyre få personer, som forstår betydningen af standardisering af dokumentformater. De krav, som fx. US Defense stillede til leverandører af F16 flyet, om at dokumentationen skulle være til rådighed for elektronisk tilgang/læsning, og at den skulle være søgbar og skulle kunne skrives ud på forskellige printere og andre devices, de krav er stort set ukendte i Danmark. Vedligehold af dokumentationen gennem flyets levetid (30 år) ville være vanskelig eller umulig uden et SGML system.

DKuug er selvfølgelig en lille brik i det store spil om standarder, ligesom vores mand i Rådet for Større IT-sikkerhed er en ud af mange. Naturligvis. Men har skubbet på og vi bliver ved.

Igennem årene viste det sig, at initiativerne flyttede sig ud af DKuug, til SSLUG, BSD-DK, Coord (Linuxforum) m.v. DKuug har støttet de store arrangementer, som fx. SSLUG afholdt: Linux-forum og Open Source Days. Det var ikke initiativer fra DKuug, der skabte de konferencer. At sidde i bestyrelsen i DKuug er ikke prestigiøst og skal heller ikke være det, men DKuug har en opgave i at forvalte de ressourcer, som vi har været lige ved at miste og bruge dem til gavn for FREE software, åbenhed og vidensdeling.

■

Indhold i dette nummer

DKuug er både de gamle og de nye medlemmers forening. Vi har gennem de sidste 3 år prøvet at øge troværdigheden, vi har arbejdet med forskellige retningslinier for indhold og er endt med gode artikler fra mange sider, som regel. Der skal være noget om hardware, software og nye programversioner i hvert nummer, foruden politisk stof og foreningsnyt.

Det vigtigste er, at stoffet kan læses af både begyndere og viderekomne, og det begrænser de faglige perspektiver en hel del. Da jeg skrev mit bidrag til dette nummer af bladet, stødte jeg igen på et problem: De unge Unix - eller Linux brugere, for nu at være ærlige - har ikke meget erfaring med installation fra source. For en del mennesker, også pakke-maintainere, opfattes det som et problem at installere fra source. I min optik er det omvendt: Pakke-vedligehold tilføjer et lag af komplexitet, og dermed en fejlkilde. Programmer fra source, som kompilerer uden problemer (uden ændring af source) har under byggeprocessen med ./configure selv checket, at de til rådighed værende supportlibraries fungerer.

Der er systemer som BSD og Gentoo, der anbefaler installation fra source - af forskellige grunde, men dog med samme resultat. Installation (rebuild) kan tage lang tid. Men der er også fordele ved denne fremgangsmåde. Det vil vi tage op i kommende artikler.

Da en ny version af det mest udbredte fuldt implementerede *helt* Open Source RDBMS PostgreSQL kom under luppen i dette nummer, fandt vi det nødvendigt at tage læserne ved hånden gennem byggeprocessen.

PostgreSQL (og MySQL m.fl.) er omfattende systemer, men vi har alligevel skønnet det nyttigt at give læserne en hands-on oplevelse, som kan skærpe appetitten for udvikling.

I dette nummer har vi igen en artikel af David Askirk om et HW-projekt, denne gang på Arduino. Det afspejler lidt af hvad der foregår i Labitat.dk som vi så i år på OSD; en forening, som DKuug også gerne støtter.

Donald Axel

DKuug-NYT er medlemsblad for DKuug, foreningen for Åbne Systemer og Internet
Nr. 172 - Oktober 2013

Udgiver:
DKUUG
Fruebjergvej 3
2100 København Ø
Tlf. 39 17 99 44
email: dkuugnyt@dkuug.dk
Redaktion:
Donald Axel (ansvarshavende)

Forsidecredits:
(redaktionen)

Design og layout:
DKUUG/Donald Axel
Annoncer:
pr@dkuug.dk
Tryk:
Lasertryk i Aarhus
Oplag:
500 eksemplarer

Artikler og inlæg i DKUUG-Nyt er ikke nødvendigvis i overensstemmelse med redaktionens eller DKUUGs bestyrelses synspunkter.

Eftertryk i uddrag med kildeangivelse er tilladt.

Deadline for nr. 173: 10. November 2013

Medlem af Dansk Fagpresse
DKUUG-Nyt
ISSN-1395-1440

Vores møder og **foredrag** holdes - med mindre andet udtrykkeligt angives - på vores adresse:

DKUUG
SYMBION
Fruebjergvej 3
2100 København Ø

Hvis man kommer lidt før, er der tid til en snak på kontoret. DKUUG bor i en virksomhedsfarm, Symbion, hvor der er åbne døre indtil kl.18. Efter den tid har vi på foredragsaftener en vagt ved døren.

INDHOLD:

Arrangementer:

DKUUG fødselsdag: D. 18 november 2013 har foreningen eksisteret i 30 år.
Det markeres med åbent hus på kontoret i Symbion.

Bent Bagger
Linux firewalls: Hvordan netfilter og iptables/ip6tables virker,
24 oktober i Symbion kl. 19.

Andre arrangementer vil blive annonceret på web og via mail.

Gør-det-selv foredrag:

Kom og få dine kompetencer plejet - hold et foredrag eller workshop om det, der interesserer dig.

Vores kontor i Symbion fungerer godt; vi afholder gruppe-arbejde og kan afholde små kurser og workshops, både dag og aften. Skriv til pr@dkuug.dk eller til bestyrelsen i DKUUG, bestyr@dkuug.dk, og hør om lokalet er ledigt den dag du vil arrangere et møde. Der er hurtig internetforbindelse, både wired og wireless. Er der større tilmelding, kan vi leje mødelokaler i Symbions mødested. Spørg os!

Deadline for DNyt nr. 173: Søndag d. 10. novmber 2013.

DKUUG-Nyt for 29 år siden

Medlemsblad for Dansk Unix-system Bruger Gruppe

DKUUG NYT

NR.1, OKTOBER 1984

Redaktion: Carsten Reimers, BKI
Bo Svarre Nielsen, Hewlett Packard
Kim-Biel Nielsen, Metric A/S

Adresse: DKUUG NYT ved Carsten Reimers,
Beton og konstruktionsinstituttet,
Elektrovej, b. 371, 2800 Lyngby.
Tlf: (02) 88.66.22

Så er første nummer af foreningens blad endelig en realitet. Det er denne gang forfattet udelukkende af redaktionen, men i de følgende numre vil vi introducere en række faste rubrikker, som er til medlemmernes frie disposition. Vi håber at I benytter disse flittigt, således at bladet sammen med medlemsmøderne kan gøre DKUUG til en levende forening.

INDLÆG TIL BLADET

Medlemmer, som har noget på hjerte, opfordres til at indsende indlæg til bladet. Har du gjort interessante opdagelser eller haft store vanskeligheder under brugen af Unix, så lad andre blive delagtiggjort i dine tanker. Eller har du en god ide til danske applikationer, så skriv om det.

Første nummer udkom et år efter stiftelsen

De første numre viser en foreningens en forening igang med at konstituere sig. De mange stiftere af DKUUG så, at Unix var et håb om bedre og mere standardiseret software, en oplagt måde at effektivisere. Men der var ikke nationale tiltag i retning af standardisering af styresystemer på dette tidspunkt.

Unix var inspiration for MSDOS især omdirigering af program in/output og filsystem-struktur med subdirectories.

DKUUG NYT NR 1 3 OKTOBER 1984

GAVER TIL FORENINGEN

Foreningen har modtaget et 300 baud og et 1200 baud modem som gaver fra henholdsvis Metric A/S og Selskabet for Rationel Almen Planlægning (RAP). Vi takker for denne storsindede gestus og gør samtidig andre medlemmer opmærksom på denne mulighed.

UDLEVERING AF MEDLEMSLISTER

Bestyrelsen har overvejet at udlevere listen over foreningens medlemmer til alle eventuelt interesserede. Medlemmer, som ikke ønsker at optræde på en sådan liste, kan blive slettet ved henvendelse til bestyrelsen.
Listen udleveres naturligvis til alle medlemmer.

METRICS OPKOBLINGSTILBUD

Medlemmer, som ikke har adgang til et Unix system, har nu mulighed for at koble sig ind på Metrics Zilog maskine Kontiki og få lidt "fingeren på tasten" erfaring med Unix. Kontiki er koblet til det danske Unix net.
Interesserede kan henvende sig til Kim-Biel Nielsen hos Metric på (02) 80.42.00.

DIKU UNIX-SERVICE

Har du problemer med Unix, kan du prøve at henvende dig til DIKU's Unix-service. Ring til (01) 83.64.66 og spørg efter edb-afdelingen. Måske kan en af medarbejderne her klare dit problem.

Side 3 i første nummer af DKUUG-nyt indeholder bl.a. meddelelse om 1200 baud modem modtaget som gave

DKUUG NYT NR 2 6 marts 1985

Nøddeknækkeren

Bladet har modtaget denne lille stump kode til at klare et ofte set problem. Bidragsyderen er Kim Biel-Nielsen, Metric.

BAUD RATE PÅ SERIEL PORT

```
/*********************************************************************/
/*  RE: PROGRAM TO CHANGE A SERIAL PORT'S BAUD RATE AND HOLD IT      */
/*********************************************************************/
/*  This program opens a serial port, sets the baud rate to 1200,    */
/*  then forks a background listener. Change the baud rate within    */
/*  the program (#define BAUD_RATE B1200) as necessary.             */
/*********************************************************************/
```

Nummer 2 kom så i 1985. I det år begyndte Data General at sælge den første bærbare lap-top agtige maskine, men med en LCD skærm, som var umulig at læse.

Medlemsblad for Dansk Unix-system Bruger Gruppe

DKUUG NYT nr.2, marts 1985

Redaktion: Carsten Reimers, BKI
Bo Svarre Nielsen, Hewlett Packard
Kim Biel Nielsen, Metric A/S

Adresse: DKUUG NYT ved Carsten Reimers,
Beton og konstruktionsinstituttet
Elektrovej, b. 371, 2800 Lyngby.
Tlf: (02) 88.66.22

Redaktionelt

'er har I så det andet nummer af "DKUUG nyt". Jojo, det var meningen, at det skulle være en tilbagevendende begivenhed.

Vi håber, at I kan lide det nye format på bladet - det er Indre By-terminalens Agfa P400 laserprinter, der har lagt ryg til.

I dette nummer indbyder vi til to **medlemsmøder den 18. april og den 20. juni.** Vi har nyt fra EUUG, mere om EUUG konferencen 11.-13. september i København, der er diverse nyheder om standardiseringskomitéen, UNIX-nettet, DKUUGs boggruppe og skrifter der kan rekvireres fra DKUUG, samt et godt tilbud på tidsskriftet "PC world". Sluttelig har vi referatet af generalforsamlingen d. 18. nov. 84, samt et lille program til at sætte en printerport op.

Der er vedlagt en ny medlemsliste. Husk at listen ikke er til kommerciel brug.

Endvidere har vi tilladt os at vedlægge en faktura for medlemsskab 85, vær sød at betale den hurtigt.

DKuug orienterede ved møder om standardisering af Unix - det, som blev til Posix-standarden. US Defense (og andre) krævet, at IT systemer skal være Posix conformant. MS-NT-4 (ca. 8 år senere) er i stand til at opfylde kravene stillet i Posix.

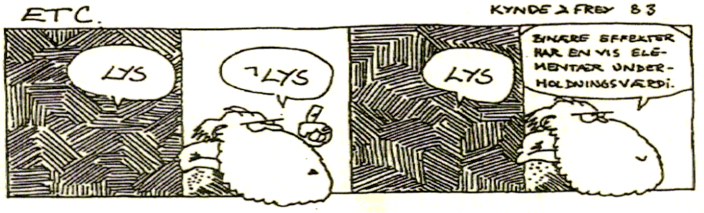
Der var også et programmeringseksempel i nr.2, et lille program til at sætte modem-hastighed. Netværk var dengang via modem for langt de fleste installationer.

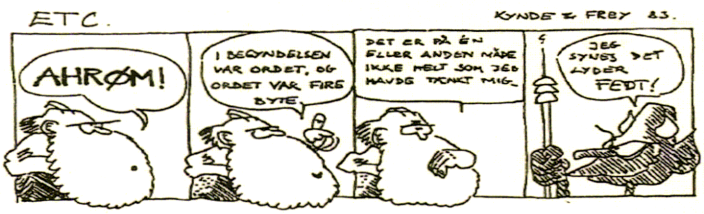
10 år efter

I 1993 skrives der om åbne systemer, om objektorientering og om X11, Grafisk userinterface til Unix. Rundt om i virksomhederne i Danmark er PC'en ved at komme på banen med Albatros, Concorde, PC-Plus og nogle andre ting. Onde tunger sagde at Albatrossen havde svært ved at lette og at Concorden, overlydsflyet, allerede var forældet.

X11, derimod, er ikke blot ved at vinde frem, men skibsværfter, lægevidenskab, meteorologi, consumer-design af forskellig slags har taget Unix tilsig, først og fremmest Silicon Graphics systemer, men også Data General og andre. En enkelt workstation kan let koste over 100.000 kr. Radar-ingeniører har udviklet specielle skærme med meget høj opløsningsevne. PC'erne begynder at kunne køre MS-Windows-3.1, som kan multitaske, men ikke preemptive - programmer skal være skrevet specielt til Microsoft platformen.

En Unix-programmør, D.J.Delorie, har porteret Gnu C-compileren til DOS, og den kører i proteced mode og kan udnytte Intel-386 arkitekturen til fulde. Med den kan en del Unix-programmer køre på MS-Dos og brugere får mulighed for at stifte et positivt bekendtskab med Open Source. Linux v.0.99 kører!





Fra nr 65, 1993:

Fra missionærer til mainstream

En ufatteligt seriøs forening fylder 10 år

journalist Ole Farbøl

"Der må da for pokker i løbet af 10 år have været én af jer, der er trådt ved siden af, har begået en heroisk katastrofe, eller på anden måde have bidraget til underholdningen!"

Scenen er sat hos UNIWARE en solrig mandag - dén dag det ikke regnede i efteråret 1993 — og Kim Biel-Nielsen vrider sig i stolen, mens han overvejer udsagnet og spoler historien tilbage til DKUUG's spæde start.

Ti år er lang tid i UNIX-historien, men alligevel giver han i første omgang op og slår afværgende ud med armene:

"Vi har været en ufatteligt seriøs forening", siger han nærmest undskyldende.

Og mens tavsheden et øjeblik sænker sig over Vedbæk, slår deres udsendte medarbejders hjerne pr. ren automatik omgående op på den side i den mentale ordbog, hvor man kan finde ord som dødbidere og vantro.

Der var alligevel engang...

Det signal kan Kim Biel-Nielsen tilsyneladende læse, for spørgsmålet koger åbenbart videre, og som de hedengangne år passerer revy på hans mentale skærm, viser det sig, at der alligevel var engang...

"Da vi i 87-88 skulle til møde i Århus, var vi så mange i Danair-maskinen, at SAS annoncerede mødets praktiske detaljer i flyet. Vi var en hel bunke UNIX-leverandører.

"det lige før, at stenalderen og 1983 var to begivenheder, som fulgte lige i halen på hinanden"

Da mødet var slut, kastede de deltagende IBM'ere, som var helt nye på UNIX, sig over nogle nye kundeemner, som de havde pin-pointet under mødet. De har jo sans for det merkantile, men det kneb med busafgangen.

Nogle artikler handler om kampen om markedet. Vil Unix kunne vinde over Microsofts WindowsGUI?

en UNIX-løsning.

Desuden kan der spares penge på hardware-anskaffelsen. Det er således de kortsigtede mål, der er de vigtigste.

På langt sigt er en UNIX-løsning med netværk og PCere ikke nødvendigvis så billigt som først antaget, da der kræves en del menneskelige ressourcer til drift af disse systemer.

"Åbne Systemer med både teknik og visioner"

Undertegnedes personlige opfattelse er dog at disse omkostninger vil blive reduceret efterhånden som der til UNIX kommer systemer til automatisk driftsovervågning svarende til de, der kendes fra mainframe-verdenen.

UNIX markedet har i dag en stor vækst indenfor workstations og mindre UNIX-maskiner. Derimod går det trægt med salget af de heltstore UNIX-systemer.

På trods af "truslen" fra Microsoft med Windows NT har UNIX stadig et langt liv foran sig.

Med det seneste tiltag, COSE, står UNIX rustet til at tage kampen op. Med COSE får UNIX nu en fælles grafisk brugergrænseflade samt standardiserede hjælpeværktøjer. En kort, præcis beskrivelse af COSE kan bedst udtrykkes med følgende citat af Brian Eberhardt:

"PC-verdenen har haft dette de sidste fem år, men nu har vi det altså også på UNIX."

PC verdenen

Åbne Systemer er ikke kun UNIX. Det var derfor interessant at høre om fremtiden for "verdens mest åbne platform", PCeren.

At PCeren bliver kraftigere, billige og mere fleksibel med næsten ubegrænsede muligheder kom ikke som et chock for nogen af tilhørerne.

Alligevel var det interessant at høre hvor langt man allerede var idag. Ikke mindst den store mobilitet indenfor PC-verdenen med trådløs kommunikation giver nye muligheder.

Dagens lille overraskelse var vel præsentationen af Wabi, der gør det muligt at afvikle MS Windows-applikationer på UNIX-arbejdsstationer. Med Wabi kan man benytte standard Windows-applikationer uden at skulle investere i hverken DOS- ellerWindows-licenser.

Konklusion

Seminaret var delt op i markedet, konsortierne, teknologi samt fremtiden. Man fik således dækket mange sider af "Åbne Systemer" med både teknik og visioner. Tilbagemeldingen fra tilhørerne var da også gennemgående meget positiv.

DKUUGs Medlemsmødeudvalg håber at se endnu flere deltagere til næste års seminar, hvor der igen vil blive arrangeret "Markedet for Åbne Systemer — nu og i fremtiden".

Åbne systemer er tydeligvis en hjertesag for DKuug anno 1993

Tyve år efter

145 / august 2003

Organizing Users - Comparing Danish Linux, BSD, and UNIX Groups

Kenneth Geisshirt
knetH@geekclub.dk

UNIX, Linux and BSD users have always understood the importance in meeting regularly with other users in the same geographical region. This paper will examine the different approaches followed by UNIX (DKUUG), Linux (SSLUG and FLUG) BSD (BSD-DK) users in Denmark. A number of success criterias will be proposed, and the success of the User Groups will be measured and presented. Moreover the paper will examine the area of mutual interest and how the User Groups coordinate their works.

Introduction

The UNIX, Linux and BSD (primarily FreeBSD but to some extend also NetBSD and OpenBSD) are well-established computing platforms in Denmark. Moreover, the user groups primarily operate in the same geographical area.

The Danish UNIX User Group, DKUUG, has its headquarter located in Copenhagen. The focus of DKUUG is open systems in general and any UNIX in particular. In the early years commercial unices dominated, and with a number of UNIX vendors as members DKUUG is the only professional driven user group in this study - professional in the sense that DKUUG has an office and a paid staff.

The BSD-DK user group is a user group focusing on the operating systems derived from the source code known as Berkeley Software Distribution (BSD). The group is for all Denmark but in reality it is situated in Copenhagen.

Fyn Linux User Group, FLUG, has its home in Odense, a small university city on the island Fyn. FLUG covers, as the name

Bladet opfattes som en udkigspost for IT-verdenen. Der er stadig mange store virksomheder, som deltager.

DKuug bestyrelsen og formand Myanne Olesen havde ommøbleret på foreningen, så at den var en slags paraply-organisation for SSLUG og BSD og andre aktive foreninger. Linuxforum blev drevet af en selvstændig "coord" gruppe. Linux var i 2003 tidspunkt stabil, og de fleste IT-folk var klare over, at en Apache server på et Linux system var en konkurrencedygtig løsning. Microsoft bliver bedre af konkurrencen, siger Linus Torvalds, som har slået sig ned i Californien og har fået fast arbejde som leder af Linux-projektet, der sponsoreres af mange firmaer, bl.a. IBM.

DKUUG
Vejen til viden om Åbne Systemer og internet

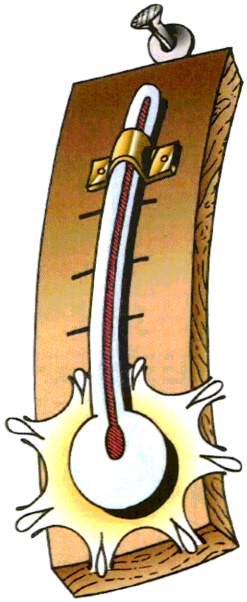
nyt

145 / august 2003

LinuxForum 2003

Organizing Users

Grid-Computeren



145 / august 2003

DKUUG-Nyt er medlemsbladet for DKUUG, foreningen for Åbne Systemer og Internet.

Udgiver:
DKUUG
Fruebjergvej 3,
2100 København Ø
Tlf: 39 17 99 44
Fax: 39 20 89 48
email:
dkuugnyt@dkuug.dk
Sekretariatet er åbent:
Mandag - fredag
kl. 10.00 - 15.00

Redaktion:
Hanne Vilmann
(ansvarshavende)
Keld Simonsen
Henrik D. Kramhøj

Tryk:
Pallas Print

Annoncer:
Kontakt DKUUGs sekretariat

Oplag:
1506 eksemplarer

Artikler m.v. i DKUUG-Nyt er ikke nødvendigvis i overensstemmelse med redaktionens eller DKUUGs bestyrelses synspunkter.
Eftertryk i uddrag med klar kilde-angivelse er tilladt.

Deadline:
Deadline for næste nummer nr. 146 er den 31. august 2003

Medlem af Dansk Fagpresse

DKUUG-Nyt
ISSN 1395-1440

FMK

Indhold

Leder

En aktivist er ifølge "Politikens NU DANSK med etymologi" en person som er medlem af en bevægelse eller en organisation der vil skabe opmærksomhed om en sag gennem forskellige former for direkte aktion.

DKUUG består af frivillige aktive...hmmm nej
DKUUG består af frivillige.....hmmm ja
DKUUG består af aktive.....

Hvor er alle de aktive henne ?
Det er en lille gruppe på få personer som driver hele foreningen.
Er tiden vokset fra DKUUG ?
Er det kedeligt at være aktiv i DKUUG ?

Det skal afslutningsvis bemærkes, at interessen for foredrag og OSD-2013 (nyt navn for Linuxforum) er uformindsket - derfor mener vi i bestyrelsen at DKuug stadig har en mission.

Donald Axel

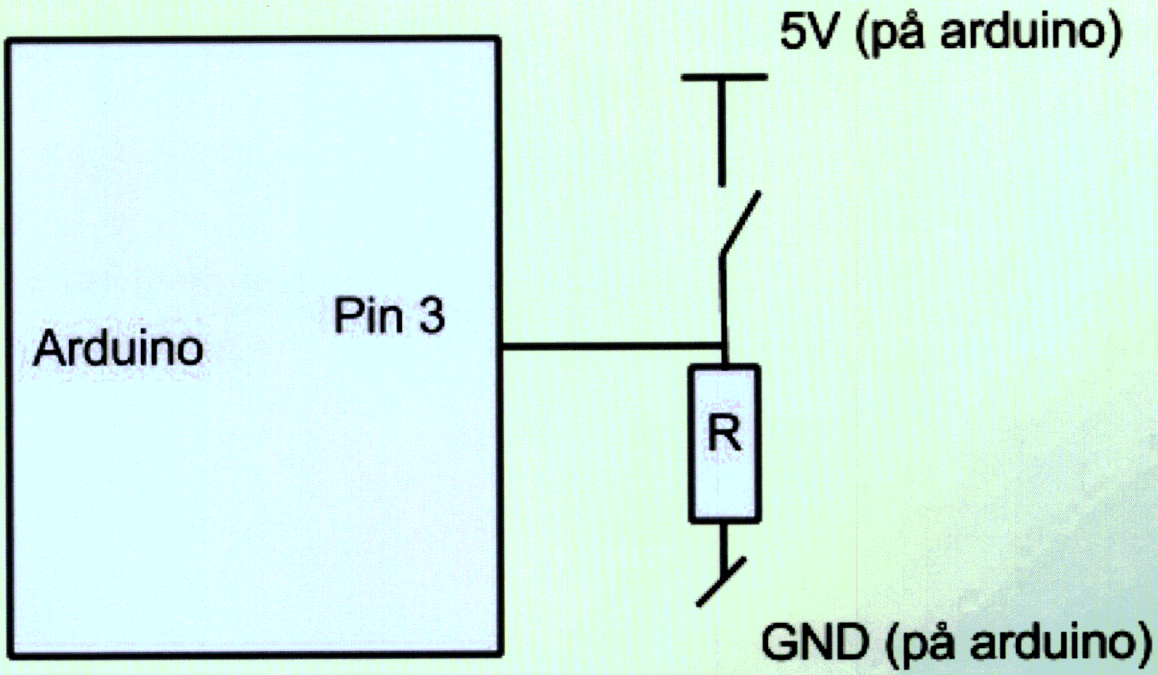
Python dørvogter

En nyttig opgave, som kan løses med Python programmeringssproget samt arduino

af David Askirk Fotel

I dag skal vi lave en lille controller til en dør. Den kan se om døren er lukket eller åben samt styre en led på en arduino. Arduinoen vil være sluttet til en computer via et USB kabel, og på computeren vil vi køre et python program.

Diagrammet:

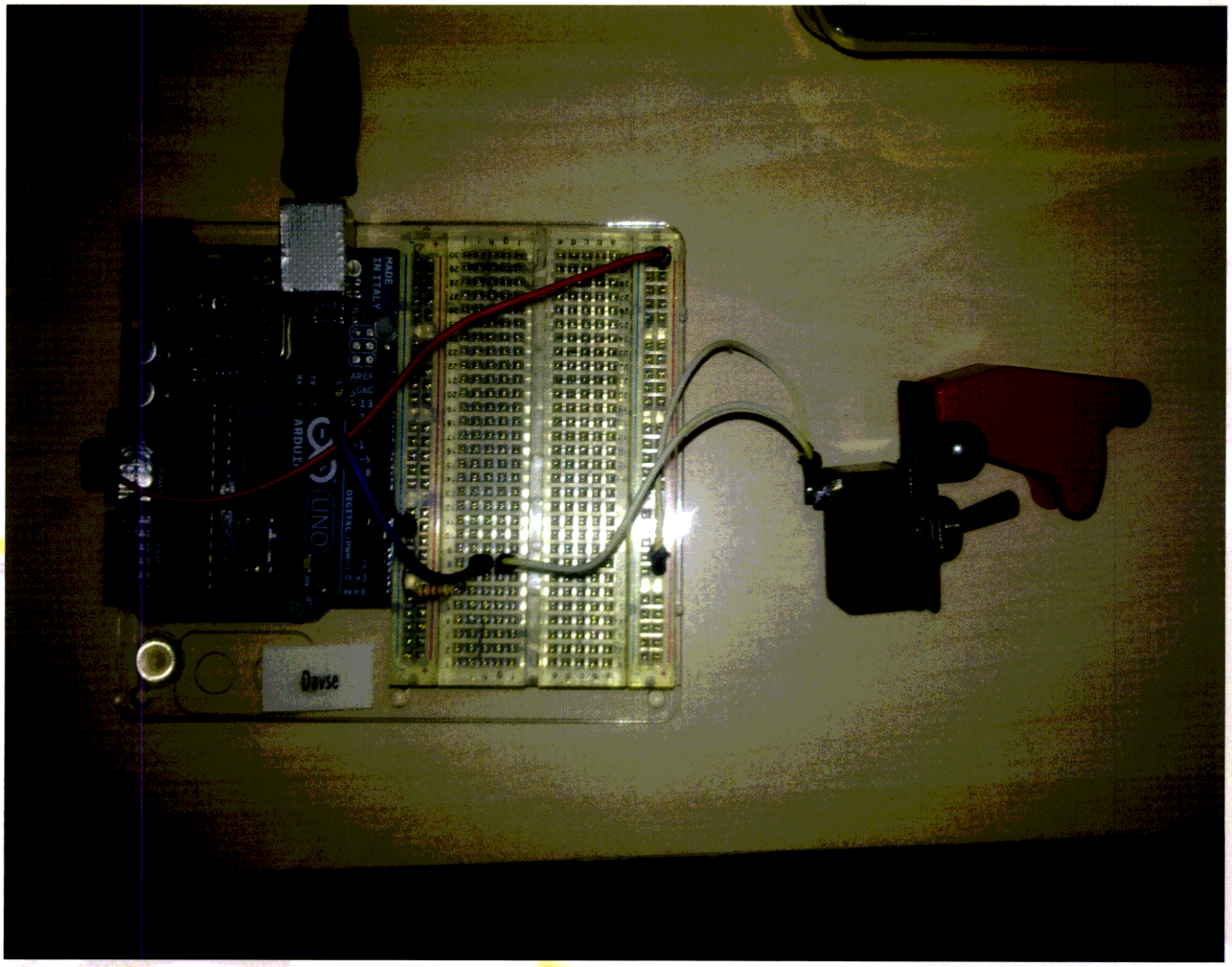


Sådan skal det forbindes. Den store firkant er arduino boardet. Fra pin 3 går det til en modstand og en kontakt. Modstanden går mellem jord til kontakten, og på den anden side af kontakten er der 5V. Dette gøres for at der altid på pin3 er enten høj (5V) eller lav (Jord). Porten (Pin3) kan ikke "svæve".

Et Arduino board består af en Atmel - en lille microcontroller. Der findes også Arduino boards med ARM 32-bit processor, men det er overkill til små projekter. En vigtig del af Arduino er standard måden, som konnektorerne er lavet, så boardet kan forbindes til forskellige udskiftelige moduler. Add-on moduler kaldes *shields* eller på dansk (somme tider) skærme. Nogle *shields* kommunikerer direkte med Arduino boardet via konnektorerne, mens andre er adresserbare via en I^2C seriel bus, som bruges til at forbinde lavhastigheds periferi-enheder til et motherboard, typisk på embedded systemer og mobiltelefoner. Derved kan mange *shields* stakkes og bruges samtidig. De officielle Arduino har brugt megaAVR serien af chips, specielt ATmega8, ATmega168 osv. De fleste boards har en 5 volt lineær regulering, og 16 MHz crystal oscillator.

Et Arduino microcontroller board er også forsynet med en boot-loader, som gør det nemt at uploade programmer til on-chip flash memory, hvor andre boards typisk tidligere behøvede en extern udviklingsenhed til at loade og køre programmer på boardet.

Her er et billede af hvordan det er forbundet i virkeligheden:



Dørkontakten fungerer på den måde, at der mellem 5V og GND sidder en kontakt og en modstand. Når kontakten ikke er sluttet, går strømmen mellem Pin 3 på arduinoen gennem modstanden til GND. Dette giver et logisk 0 som input. Når kontakten er sluttet, dvs. når døren er lukket, går strømmen fra 5V til pin 3. Dette giver et logisk 1 som input. Hvis man gerne vil bytte på om hvornår der er logisk 1 og logsik 0 kan man bytte om på kontakten og modstanden.

Ved at læse Pin3 (fra programmet) kan vi se om døren er lukket eller åben.

På arduionens ben 13 sidder der en LED (Light Emitting Diode). Den kan bruges til at vise om døren er lukket eller åben uden at kigge på computeren.

Arduino koden:

Arduino kode har som standard to funktioner. En loop og en setup. I setup bliver systemet sat op, pins bliver sat til output eller input og serial forbindelsen bliver sat op.

```
void setup()
{
        Serial.begin(9600);
        pinMode(3,INPUT);
        digitalWrite(3,HIGH);
        pinMode(13,OUTPUT);
}
```

Her bliver der sat en serial forbindelse op, som er den vi vil bruge til at snakke med arduinoen. Desuden bliver nogle pins sat til input eller output.

Her bliver der lavet et lille trick:

```
pinMode(3,INPUT);
digitalWrite(3,HIGH);
```

Først sætter vi pin 3 til input og så skriver vi til den. Dette er et lille trick, der gør at man aktiverer en intern pull-up modstand. Dette gør, at selvom man ikke forbinder noget til den pin, "svæver" pinnen ikke. Man kan derved altid vide at hvis man ikke forbinder noget, vil der altid være logisk 1 at læse på pinnen.

Pin 13 på arduinoen bliver sat til output. Det er sådan, at på et Arduino board er der på pin 13 også monteret en LED, og den kommer vi til at bruge som output fra vores arduino program.

Main metoden - while true

Resten af Arduino programmet er selve loop metoden, som bedst kan beskrives som en main metode med et while (true) loop inde i. Inde i selve while loopet, kører den kode der er skrevet i loop metoden.

```
void loop()
{
  if (Serial.available() > 0)
  {
    inByte = Serial.read();
    if (inByte == 115)
    {
      //Return status on door
      buttonState = digitalRead(3);
      if (buttonState == LOW)
        Serial.print("CLOSED\n");
      else
        Serial.print("OPEN\n");
    }
    else if (inByte == 108)
    {
      if (ledOn)
      {
        digitalWrite(13,LOW);
        ledOn = false;
      }
      else
      {
        digitalWrite(13,HIGH);
        ledOn=true;
      }
    }
  }
}
```

Her bliver der brugt serial kommunikation. I denne linje:

```
if (Serial.available() > 0)
```

Venter vi på at der er noget data klar til at blive læst.

Derefter har vi en if elseif else sætning der kigger på det data der kommer ind. Hvis det er et lille s (115 I ascii) eller et lille l (108 I ascii) så gør vi noget.

Hvis det er et lille s læser vi status på pin 3, der hvor vores kontakt er sat på. Denne kigger på om pin3 er høj eller lav. Enten returnerer vi "OPEN" eller "CLOSED".

Hvis det er et lille l der kommer ind, så kigger vi på om LED'en på pin 13 (Den er på selve arduionen) er tændt, og skifter status på den, samt på den globale status variabel.

Arduinoen er sat til en computer via et usb kabel. På computeren kører der et python program.

Det kan køre på to måder. Enten kan man styre døren via et simpelt web-api eller via terminalen.

Funktionskode til programmet følger på de næste sider:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 20 11:27:55 2013

@author: david
"""
#Bottle webframework
from bottle import route, run

#For argv
import sys

#JSON
import json

#We need to import serial so we can communicate with the arduino
import serial

#Lets make the connection to the arduino
ser = serial.Serial("/dev/ttyACM0")
ledOn = False

#Get the door status from the arduino
def get_door_status():
    ser.write("s")
    status = ser.readline()
    status = status.replace("\n","")
    return status

def light_the_led():
    global ledOn
    if not ledOn:
        ser.write("l")
        ledOn = True

def led_off():
    global ledOn
    if ledOn:
        ser.write("l")
        ledOn = False

def get_command():
    input_command = raw_input("Enter command: ")
    if input_command == "S" or input_command == "s":
        return "STATUS"
    elif input_command.lower() == "q":
        return "QUIT"
    elif input_command.lower() == "l":
        light_the_led()
    elif input_command.lower() == "o":
        led_off()
    else:
        return "NONE"

def print_usage():
    print "Door control system"
    print "(S)tatus of the door"
    print "(L)ight the led"
    print "(O)ff the led"
    print "(Q)uit"

#Status of door
@route('/')
def index():
    return json.dumps({'door_status':get_door_status()})

@route('/led_on')
def led_on():
    light_the_led()
    return json.dumps('OK')
```

```
@route('/led_off')
def turn_led_off():
    led_off()
    return json.dumps('OK')

if __name__ == "__main__":
    command = "NONE"
    if len(sys.argv) == 2 and sys.argv[1] == "web":
        run(host='localhost', port=8080)
    else:
        while not command == "QUIT":
            print_usage()
            command = get_command()
            if command == "STATUS":
                print "*"*20
                print "Door is", get_door_status()
                print "*"*20
    ser.close()
```

I starten af programmet importerer vi nogle biblioteker der skal bruges:

```
from bottle import route, run
import sys
import json
import serial
```

Udover hvad der ligger i standard biblioteket til python, bruger vi ***Bottle*** **(http://bottlepy.org/docs/dev/)** et micro webframework og pySerial **(http://pyserial.sourceforge.net/)**

Efter importeringer sætter vi serial forbindelsen til arduino op; derfor er det vigtigt at den er forbundet til computeren når programmet startes.

```
ser = serial.Serial("/dev/ttyACM0")
```

Det kan ses at vi bruger porten /**dev/ttyACM0** - på lidt ældre arduinoer hedder den /dev/ttyUSB0. Dette er fordi producenterne bag arduino har skiftet USB chip, så den bruger en anden driver på Linux.

Dernæst følger der nogle funktioner til at hente dørstatus ud, samt tænde og slukke på LED'en.

```
def get_door_status():
    ser.write("s")
    status = ser.readline()
    status = status.replace("\n","")
    return status
```

I denne funktion henter vi dør statusen ud. Vi skriver 's' til arduinoen og venter på svar. Dette svar, som enten er OPEN eller CLOSED, returnerer vi som status på døren.

```
def get_command():
    input_command = raw_input("Enter command: ")
    if input_command == "S" or input_command == "s":
        return "STATUS"
    elif input_command.lower() == "q":
        return "QUIT"
    elif input_command.lower() == "l":
        light_the_led()
    elif input_command.lower() == "o":
        led_off()
    else:
        return "NONE"
```

Denne kan få en kommando fra command line ud. Den tester på to forskellige måde. I første linje samligner vi først om det er et stort S eller et lille s. I de andre linjer bruger vi et python trick til at lave det om til et lille bogstav inden vi sammenligner.

Nu kommer der til de få funktioner det er nødvendigt at bruge for at benytte sig af bottle:

```
@route('/')
def index():
    return json.dumps({'door_status':get_door_status()})
```

Her defineres først at det er på den route der hedder / altså index siden, skal den køre den metode der returnere dør statusen.

De andre to tænder og slukker LED'en.

I den sidste del af python programmet styrer den om det er et web interface eller et commandline interface der skal bruges:

```
if __name__ == "__main__":
    command = "NONE"
    if len(sys.argv) == 2 and sys.argv[1] == "web":
        run(host='localhost', port=8080)
    else:
        while not command == "QUIT":
            print_usage()
            command = get_command()
            if command == "STATUS":
                print "*"*20
                print "Door is", get_door_status()
                print "*"*20
    ser.close()
```

Denne del af koden bliver kun kørt hvis man kører selve python filen. Dette er styret af denne linje:

```
if __name__ == "__main__":
```

Dette er et godt trick til at lave små test muligheder i sine klasser hvis man gerne vil kunne køre lidt test direkte i python filen, men stadig vil bruge klassen ved fx. en import kommando.

Ellers læses der fra argv, for at se om der er skrevet web eller ikke er skrevet noget som første argument til programmet.

I det her eksempel styrer vi blot en LED på en Arduino. Men dette kunne også være et relæ så man kunne styre fx. en dør lås.

Arduinoen har gjort det nemt at komme i gang med elektronik, og python gør det nemt at skrive programmer på ens computer til at styre en Arduino. Mulighederne er uendelige.

PostgreSQL 9.3

giver anledning til at se nærmere på Open Source databaseprojekter

af Donald Axel

På postgresql.org websitet kunne man 9. september læse følgende: ***PostgreSQL 9.3 released!***

> *The PostgreSQL Global Development Group announces the release of PostgreSQL 9.3, the latest version of the world's leading open source relational database system. This release expands PostgreSQL's reliability, availability, and ability to integrate with other databases. Users are already finding that they can build applications using version 9.3 which would not have been possible before.*
>
> *http://www.postgresql.org/about/news/1481/*
>
> *"PostgreSQL 9.3 provides features that as an app developer I can use immediately: better JSON functionality, regular expression indexing, and easily federating databases with the Postgres foreign data wrapper. I have no idea how I completed projects without 9.3," said Jonathan S. Katz, CTO of VenueBook.*
>
> ***Writable External Data***
>
> *Version 9.3 makes PostgreSQL's* ***Foreign Data Wrappers*** *writable, enabling two-way data interchange between systems. Today's complex IT environments involve multiple databases and semi-structured data sources, and PostgreSQL helps you integrate them into a coherent stack. The project has also released postgres_fdw, a higher-performance, read/write PostgreSQL-to-PostgreSQL federation driver.*
>
> ***"Writable foreign data wrappers enable us to plug in and seamlessly test various backend alternatives, allowing us to address different needs quickly and prototype intelligently," explained Lee Holloway,*** *Co-founder and Lead Engineer at CloudFlare. "It is exciting to conceive and toss up new data stores (including our in-house experimental ones written in Go) and then watch them read, write, and even transact each other."*
>
> *(Redaktionens fremhævelser).*

Uagtet at det er stort, at PostgreSQL er vokset fra et system plaget af fejl og mangler til et system, som tillader sammenkobling af egne **og andre leverandørers** databaser, så blev forfatteren af nærværende artikel mødt med den klassiske kommentar: Ja men vi har jo også MySQL, og den er (eller var) hurtigere. Min erfaring med at bygge løsninger baseret på databaser er minimal, og omfatter ikke løsninger i millionskala. Til gengæld har jeg været i nærheden af Danmarks første 4 TB, 24-maskiners parallel-cluster DB2 database for telefon-forbrug for 16 år siden, og har siden dengang foretaget adskillige forsøg med simulerede data på databaseplatformene DB2, Oracle for Linux, MySQL, MySQL/InnoDB og - selvfølgelig - PostgreSQL (i daglig tale kaldet Postgres).

Hvorfor PostgreSQL? Hvorfor ikke MySQL?

Redaktionen har ikke mødt modsigelser overfor den påstand at PostgreSQL fortjener at blive omtalt grundigt, fordi den er et elegant og meget nyttigt projekt. I denne artikel ser vi på de grundlæggende features, og senere artikler vil give en tur gennem elementær web-applikation. I et kommende nummer vil vi også se på forskellene mellem de forskellige database-systemer: MySQL, MariaDB og NoSQL og måske Linux-version af Oracle, DB2 og flere.

PostgreSQL og MySQL har begge ændret karakter genem årene. I 1999 eller 2000, da PostgreSQL kom på RedHat som "database**n**" (det var før *Enterprise Linux*) var PostgreSQL ikke så stabil, men til mange opgaver var den stabil nok - man prøver om det kører og tager problemerne efterhånden, når det drejer sig om mindre kritiske opgaver i en mindre virksomhed. I år 2000 kørte PostgreSQL (og MySQL) som single CPU applikationer, og datasikkerheden bestod i backup. Driftssikkerheden bestod i at man altid kunne komme igang igen efter et crash. PostgreSQL havde transactions og nested queries, men MySQL var hurtig som read-mostly backend for et dynamisk website.

I dag kan både PostgreSQL og MySQL køre cluster/parallelt, og kører stabilt, MySQL kan køre rigtig SQL og har som back-end både MyISAM og InnoDB (rigtig SQL base); men der er sket meget mere, især *organisatorisk:* I 2008 blevMySQL er købt af SUN Microsystems, som året efter blev overtaget af Oracle. Det var måske ikke det værste, der kunne ske, fordi Oracle har baseret sin forretning på standarder (PLSQL, SQL, TCP/IP, er standarder). En Oracle bruger vil nemt kunne dele sine data med en anden SQL database. Men overtagelse af MySQL betyder at nogle brugere har fået betænkeligheder. Nu for tiden gør MariaDB (et fork af MySQL) indhug i MySQL brugerskaren; således fx. **Wikipedia,** verdens sjette-største webservice, som er gået fra MySQL til MariaDB.

Hvem bruger PostgreSQL?

Både PostgreSQL og MySQL fik store kunder i begyndelsen af nullerne. De var de første store back-ends til webservices baseret på Linux og Open Source. I en liste over store Open Source brugere fra 2003 blev det nævn, at *Afilias* var bruger af PostgreSQL; Afilias var teknisk service (drift) af dot.org domænet. Sidenhen er Afilias også drift for mange andre topdomæner, bl.a. de asiatiske, og administrerer mere end 24 mio. domænenavne.

I 2008 nævnes Meteo France, det franske meteorologiske institut, som bruger af PostgreSQL; PostgreSQL har indbyggede Geografisk Informations System (GIS) funktioner.

Beskrivelsen af brugen findes på Postgresql.org, og French Meteo IT-ansvarlig Valerie Schneider spørges om nærmere omstændigheder af Jean-Paul Argudo, der skriver for French PostgreSQL:

> *J-P: Hvor mange ansatte er der i organisationen?*
>
> *Valerie: 3600 mennesker fordelt over Frankrig.*
>
> *J-P: Hvordan bruges databaserne i jeres organisation?*
>
> *Valerie: De fleste af kerne-applikationerne i Meteo France er baseret på databaser fra adskillige leverandører, teknisk og videnskabelige databaser, af hvilke nogle er real-time, så tilgængelighed og konsistens er kritisk.*
>
> *J-P: Hvor mange databaser køres på PostgreSQL?*
>
> *Valerie: Mellem 10 og 20% køres på PostgreSQL, der arbejdes på at migrere flere. For performance-målinger bruges tabeller med 130 mio. linier, hvilket betyder ca. 30 GB pr. tabel. Vores største PostgreSQL database er på 3.500 GB (3.5 Terabytes).*

Og det var som nævnt i 2008, for 5 år siden, postgresql-8.2.

Licensen - fri og gratis - er det første argument, som Meteo France giver for at bruge PostgreSQL, og hurtighed og lethed i

betjening/administration er det næste argument. Det leder til spørgsmålet om hvordan projektet finansieres.

I dag er der en række store firmaer, som **sponsorerer** PostgreSQL: EnterpriseDB, Command Prompt og Red Hat Inc. (USA), Dalibo (Frankrig) og 2ndQuadrant (UK) er top sponsorer *("platin" sponsorer)*. De *mindre* kendte firmaer er de *største* sponsorer. Det viser, hvor skævt udviklingsomkostninger for Open Source fordeles. Et projekt som PostgreSQL burde modtage støtte fra europæiske nationer, der vil pleje EU's kompetencer og konkurrenceevne. De kendte sponsorer, Google, Skype, Huawei, HP, m.fl. kommer i tredie række *("bronze" sponsorer)*.

PostgreSQL var tidligt ude med support for multibyte char, som bruges til asiatiske tegn (*scripts*, tegnsystemer, i modsætning til alfabeter) og derfor var der en del asiatiske brugere og sponsorer, fortalte Bruce Momjan (en af grundlæggerne af PostgreSQL Global Development Group, forfatter (PostgreSQL, Introduction and Concepts) ved Linuxforum 2005.

Andre eksempler på store PostgreSQL brugere er - i tilfældig rækkefølge - LastFM, McAfee, Trend Micro, Federal Aviation Authority (FAA) for kritisk information om lufthavne i USA. FAA foreskriver, at alle nye GIS projekter skal køre på PostgreSQL.

Hurtighed som andet argument - kun?

Det var ikke hurtighed, som oprindeligt tiltalte PostgreSQL brugere. Det var først og fremmest implementering af SQL (MySQL kunne fx. ikke lave queries med midlertidige tabeller, brugeren måtte opdele i flere queries og nye tabeller, men udviklingen har ændret alt dette). Ihærdighed og vedholdenhed i udvikling og hjælp med fejlfinding, konsulenthuse med indsigt, er en yderligere grund til at PostgreSQL blev brugt, selv om MySQL var så fremherskende, at der var mange, der end ikke kendte PostgreSQL.

Brugere, som nærmede sig PostgreSQL, opdagede imidlertid hurtigt, at PostgreSQL var utrolig nem at bruge, - og at installere fra source, det er stadig lige ud ad landevejen (se side 14). Ligesom MySQL, hvor man dog skal tage stilling til, om man vil installere en backend eller to eller ... Til sammenligning er Linux version af DB2 tilgængelig for en 90 dages evaluering.

PostgreSQL er blevet hurtigere med årene. Det er længe siden, at PostgreSQL var langsommere end MySQL og Oracle. De første år af PostgreSQL udviklingen gik med at skrælle overflødig kode væk og optimere. Projektet har siden starten været baseret på Relationel Database Management systemer (RDBMS). Den grundlæggende teknik udspringer af databaseforskning på University of California, Berkeley. Michael Stonebraker på Berkeley var med fra starten i begyndelsen af 1970'erne til at udvikle database-design, oprindeligt inspireret af de banebrydende skrifter af Edgar J. Codd (IBM, DB2). Mange projekter, kommercielle såvel som Open Source, udspringer af miljøet på Berkeley.

Performance målinger af forskellig slags

I sidste del af artiklen vises nogle performance målinger, noget man selv kan gentage, hvis man har et par GB fri diskplads på en ikke alt for fragmenteret disk.

Fragmenteringsprocent kan aflæses med fsck, file system checker. Hvis ens data er på disk2, vil kommandoen typisk være

```
fsck -n /dev/sdb1
```

[...]

/tmp: 262/251136 files (5.3% non-contiguous), 18271/251007 blocks

hvor -n betyder, at fsck kører read-only - det skal man bruge, hvis man kører på en mounted disk. -y betyder "yes to all", hvis der er nogle spørgsmål undervejs, som man alligevel ikke behøver at svare på (så længe man kører read-only!)

Disk hastighed fx.:

```
hdparm -tT /dev/sdb
```

Det er klart, at både antal CPU'er, chip-sæt og motherboard bus-hastigheder, disk-hastighed og hastighed på forbindelsen mellem motherboard-bus og disk-system og andre belastningsfaktorer afgør, hvor mange transaktioner pr.sekund (T/sec) man kan opnå; men ikke desto mindre kan dette tal alligevel give et fingerpeg om flere ting, især hvis man gør sig klart, hvor de enkelte faktorer ligger i forhold til dagens norm for maskiner og systemer. Fx. er de systemer, som artiklens test er foretaget på, overhovedet ikke top-systemer, og en performance på 1500 T/sec må derfor siges at være overordentlig interessant - det vil betyde at en lille server vil kunne expedere 20-30 brugere med hver 10-20 simple queries i hurtigt tempo. Læg mærke til, at jeg ikke bare dividerer tophastigheden 1500 T/sec med 30 brugere (50) og siger så kan hver bruger spørge om 50 ting.

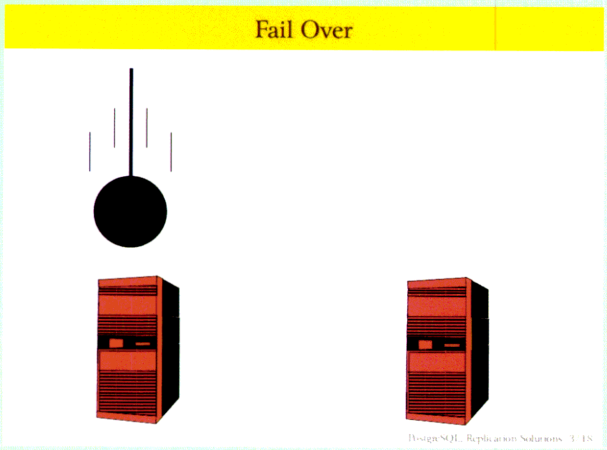
En enkelt brugerforespørgsel består ofte af 4-5 queries, nogle gange meget mere, tænk fx. på en dynamisk webside som YouTube eller Ekstrabladet. En blad-webside med fx. 150 media-elementer og 20-30 artikel-overskrifter vil foretage 30-40 databaseopslag og derefter sende videre til en front-maskine, som danner HTML og sender videre til brugeren eller en proxy (typisk Varnish skrevet af Poul Henning Kamp).

Så alt imens en moderne workstation eller en enkelt pænt udstyret rack-server kan ekspedere 10-20 aktive brugere i sekundet - svarende til anslået 600 websider i minuttet på front-end - så er der situationer, hvor en service som fx. skat eller de store el-selskaber i et land med 60-80 millioner indbyggere, har brug for services, som kan levere 100 - 200 websider i sekundet. Så får man brug for load balancing, partitioning og andre ting så belastningen kan spredes på maskiner.

Selv om maskiner kan blive lidt hurtigere, har vi nået den fysiske grænse for en del komponenter og derfor er der ikke udsigt til at man i nærmeste fremtid vil kunne sige "køb en større maskine, så løser problemerne sig". Den slags salgsargumenter har nogle af os, der har været længe i IT-verdenen, set alt for ofte, og er er inkompetent og useriøst. Et produktionssystem (offentlige services, store websteder) skal være designet med blik på balance mellem driftsomkostninger og udviklingsomkostninger, og de sidste er ofte de største, men for systemer, som forventer mange brugere, har livrem og seler ved flere lejligheder vist sig at være nødvendige. Vi nævner i flæng: Amanda, Polsag, Blå Punkt.

En tur gennem PostgreSQL fra source

På de følgende sider går vi igennem de skridt, man skal gøre for at få en fungerende PostgreSQL database med alle indbyggede features. Derefter ser vi på et minimal eksempel på forbindelse mellem flere maskiner med samme version af en PostgreSQL installation. Derefter følger en oversigt over forskellige muligheder for load-balancing, replikering og backup.



Installation af PostgreSQL fra source

At installere den nyeste version PostgreSQL (*postgres* for nemheds skyld) fra source er heldigvis nemt, og er den bedste anledning til at diskutere pakkesystemer, rettere, pro et contra vedrørende tilpasninger til distributioner. Postgres er nemlig rimelig nem at kompilere, og denne databases afhængigheder til andre systemer er til at overskue. Hvis man vil have nyeste version, er man som regel nødt til at foretage installation fra source.

I modsætning til pakkesystemer vil en installation fra source helst lægge sig i sit helt eget filtræ. Den er derfor nem at fjerne igen. En softwarepakke fra et seriøst projekt vil være meget omhyggeligt med den slags ting. Postgres installerer sig normalt til /usr/local/pgsql, og hvis man vil fjerne det hele igen, er det blot **`rm -r /usr/local/pgsql.`**

Men det forudsætter jo, at det er toppen eller nær toppen af et afhængighedstræ, dependency tree. Ellers vil der være andre programmer eller libraries med hjælpefunktioner, som er afhængige af den nyinstallerede pakke. Det gælder fx. pgadmin3, et GUI-interface til administration af postgres og PlProxy, et hjælpeprogram til database-interconnection, som vi vil se på senere i forbindelse med horisontal partitionering af data på flere servere.

Jeg har kompileret og installeret postgresql-9.3.rc1.tar.bz2 på en AMD x86_64 maskine med et 32 bit Debian 6.0.6, med 4 GB RAM ("kun", sagde en bekendt) og desuden på en Linux-Mint-13 64-bit installation (fra nøjagtig samme source).

Hvis man ikke er systemadministrator (root) kan man godt installere postgres på egen konto, forudsat at man har 250 MB til rådighed - og helst 5 GB mere til datasimulering, hvis man vil prøve store database-operationer med millioner af records, ellers er 50-100 MB nok til selve databasefilerne. Selve udviklings-filtræet ender på 150 MB, og kan jo slettes efter endt kompilering og installation, hvis man ikke regner med at kigge source efter i sømmene.

For den utålmodige:

```
./configure
gmake
su
gmake install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D
/usr/local/pgsql/data
/usr/local/pgsql/bin/postgres -D
/usr/local/pgsql/data >logfile 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

Ovenstående vil - hvis ikke der tidligere har været installeret mange softwarepakker fra source - formentlig komme med en anmodning om at man installerer readline library, fordi det muliggør kommando-interface med history og rette-muligheder.

For den videbegærlige

Det er dog altid en god idé at orientere sig om configurationsmuligheder, inden man starter gmake (som på de fleste Linux systemer bare hedder *make*).

Man kan studere options ved at køre configure-scriptet med en help option:

```
$ ./configure --help > dax-confhelp.txt
$ view dax-confhelp.txt # eller brug gvim
```

Fordelen ved at læse hjælpeteksten til konfiguration med en editor er, at man kan søge frem og tilbage efter de ord eller options, som er særlig interessante.

Parametre til configure

Her er der nogel features, som påkalder sig opmærksomhed:

```
Optional Features:
  --disable-option-checking  ignore unrecognized
            --enable/--with options

  --disable-FEATURE
      do not include FEATURE (same as --enable-
      FEATURE=no)

  --enable-FEATURE[=ARG]
        include FEATURE [ARG=yes]


  --disable-integer-datetimes
        disable 64-bit integer date/time support

[... og længere fremme:]
  --with-python
        build Python modules (PL/Python)
```

Der vil være mange features, som man ikke lige havde tænkt på - nogle af dem er forklaret i INSTALL-filen eller dokumentationen, men andre fagudtryk kan man nemt finde forklaret i Wikipedia.

Den ovenfor viste ***64 bit option for dato-support*** er interessant, for den gør jo systemet 2038-sikkert. Det er ikke sikkert, at der kører ret mange postgres-9 servere til den tid :-) men skulle man have brug for en dato efter 19.januar 2038, så har man support for det med denne option.

De fleste vil gerne have Python support, og i så fald er kommandoen for at konfigurere build-processen

```
./configure   --with-python
```

Derved bygges Python moduler som giver Python programmer acces til SQL (PL/Python). Det forudsætter naturligvis, at hele Python er installeret, og ikke kun sådan, som mange pakkesystemer gør det, uden headerfiler til kompilering af nye moduler. Desuden skal *distutils modul* være til stede. Man kan endog komme ud for, at man er nødt til at installere Python fra source for at få postgres til at bygge det sharede library *plpython.so*.

Manglende headerfiler og libraries

I denne omgang vil vi køre build processen så minimalt som muligt og vælger derfor blot at køre således:

```
./configure && make
```

Hvis et support library mangler, hvis fx. *readline* ikke er installeret, får man en fejl:

```
error: readline library not found
If you have readline already installed, see
config.log for details on the failure.  It is
possible the compiler isn't looking in the
proper directory.

Use --without-readline to disable readline
support.
```

På de fleste systemer betyder det, at man skal efter-installere den softwarepakke, som indeholder udviklingsfiler, d.v.s. på fx. Debian : apt-get install libreadline6-dev.

Hvordan finder man nu lige ud af, at det er den? Jo, de fleste systemer giver adgang til en liste over alle standard pakker, og den kan man søge i. Jeg bruger typisk at gemme den komplette liste, enten ved hjælp af aptitude eller et andet værktøj, i en fil *avail-2013-09-14* og bruger grep:

```
grep readline vail-xx | less
```

Det er ikke så nørdet, som det lyder, og har man lidt erfaring med Linux-programmering, så begynder det at blive en naturlig ting, som man ikke tænker så meget over.

PostgreSQL-*.tar.bz2 udmærker sig i øvrigt ved at have en meget god installationsvejledning, filen ***INSTALL*** som forklarer de fleste spørgsmål, man kan have, og som desuden rummer al dokumentationen. Men hvis man vil have documentation installeret i /usr/local/pgsql/share/doc, skal man faktisk huske at bygge den:

```
make world
make install-world
```

Nu er databasen klar til brug og manual sider kan læses.

Kan vi SÅ komme igang?

Men programmerne til start og administration af databasen ligger som nævnt samlet i /usr/local/pgsql/bin og det var måske ikke lige hvad man havde tænkt sig. Normalt, når man installerer en pakke, kan man jo bare bruge den med det samme.

De to løsninger, som man almindeligvis bruger, har hver deres fordele og ulemper: Symlinks eller $PATH ***(eller sti)*** tilføjelse.

Symbolske links

Den ene er at oprette symlinks, symbolske links eller shortcuts (genveje) som nogle PC-folk kalder det, fra alle filer i /usr/local/pgsql/bin til /usr/local/bin. Det forudsætter, at man kører en shell med root rettigheder:

```
ln -s /usr/local/pgsql/bin/* /usr/local/bin/
ln -s /usr/local/pgsql/lib/* /usr/local/lib/
```

og hvis man også vil gøre livet lettere for sig selv ved at manual siderne ligger tilgængelige på den sædvanlige måde, hvor man blot skriver fx. “man postgres”

```
ln -s /usr/local/pgsql/share/man/man1/*\
 /usr/local/share/man/man1/
ln -s /usr/local/pgsql/share/man/man3/*\
 /usr/local/share/man/man3/
ln -s /usr/local/pgsql/share/man/man7/*\
 /usr/local/share/man/man7/
```

Fordelen ved symlinks er at det gælder for alle brugere. Hvis man fjerner hele installationen (rm -r /usr/local/pgsql) så har man en masse knækkede links (døde links), men de kan nemt luges ud med programmet symlinks(1), (cd /usr/local/bin/ ; symlinks -d ./) Men hvis man ønsker at have flere versioner af postgres kørende samtidig (fx. ved restore af historiske databasefiler fra en device backup) så må man bruge $PATH metoden.

Sti til programfiler, PATH

Hvis det er den eneste installerede postgres, kan path-variabelen ændres således:

```
PATH=$PATH:/usr/local/pgsql/bin
```

men hvis man har installeret lokalt i sit homedir, bør den stå forrest, ændringen:

```
PATH=/home/myname/pgsql/bin:$PATH
```

PATH er en shell-variabel og kan derfor ændres pr. session, pr. bruger -- eller hvis man ændrer i /etc/profile, så gælder det for alle brugere, der logger ind efter ændringen.

I andet eksempel, pgsql i homedir, sættes den nye path forrest, fordi vi vil være sikre på at programmer, der køres, er fra HOMEdir versionen. Hvis den ikke er kompileret med indbygget shared-library path, skal man også sætte LD_LIBRARY_PATH.

For C-shell og derivater er der en anden syntax for denne setting.

Der kan siges meget mere om shell variable og programeksekvering; for den lejlighedsvise kommandoliniebruger er det ofte en kilde til forundring og fejltagelser.

Initialisering af databasefiler

```
adduser  postgres
mkdir /usr/local/pgsql/data
chown postgres:postgres /usr/local/pgsql/data
su - postgres
initdb
```

Tilføjelse af en almindelig bruger, egtl. en konto: denne konto behøver ikke at have login-name postgres, og det behøver ikke være en systembruger med en lav UID, det kan være hvilket som helst navn (fx. *dbadm, pgsql* eller hvilken som helst bruger bortset fra root, UID 0). Det er håndhævelse af princippet for *defensive programming:* postgres kan gøre alt, hvad der skal gøres, som en upriviligeret bruger, så hvorfor køre som root? Det er dog ikke nogen sikring mod alle forsøg på intrusion, der vil stadig være risiko for at en fejl i koden kan bevirke at en SQL injection kan eskalere processens privilegier og derved skabe rod.

Oprettelse af en demodatabase

Initdb opretter tomme filer, og tre databaser, template0, template1 og postgres. I disse baser ligger oplysninger for strukturering af alle kommende databaser. Initdb må kun køres én gang. Som postgres (eller det navn, vi har brugt):

```
pg_ctl start -D /usr/local/pgsql/data
```

Prøv at se, om programmet kører, fx.:

```
ps u -C postgres
```

eller

```
pg_isready
```

Det skulle gerne vise en del processer, som udgør selve database management systemet. Nu kan vi skabe en database fra kommandolinien (hvis vi vel at mærke kører på kontoen postgres!):

```
createdb testdb
```

Nu har database-programmet skabt en database, d.v.s. et tomt rum (namespace) for nye tabeller (datafiler eller registre, SQL terminologien kalder det for tabeller, ***tables***).

Derefter startes kommando interface til databasen. Programmet psql er en front end, som kan sende SQL kommandoer til back-end programmet.

```
psql -U postgres -d testdb
```

Det skulle gerne give os et promtp (kommandolinie til SQL):

```
testdb#
```

og på denne bør man nu afprøve minimum flg.:

```
testdb# \l
[viser hvile databaser, der er oprettet]
testdb# \?
[help text til backslash-kommandoer]
testdb# create database simple;
CREATE DATABASE
```

Når en kommando er udført, kvitterer postgres med et par ord, som viser at det gik godt. Det kan man undgå ved option -q kommandoen bliver så **`psql -q -U postgres -d testdb`**

løvrigt kan man udelade -U postgres hvis man er logget ind som postgres. **psql** kan meget mere; fx. kan psql programmet kommunikere med andre hosts, som tillader remote access til deres postgres database.

Forudsat vi har adgang til en server, som svarer på navnet dbserver (eller IP eller fuldt kvalificeret domænenavn, fx. dbserver.bizz.dk), kan vi tilgå postgres på dbserver på følgende måde:

```
psql -q -h dbserver -U walter -d business
```

Installation af pgadmin3

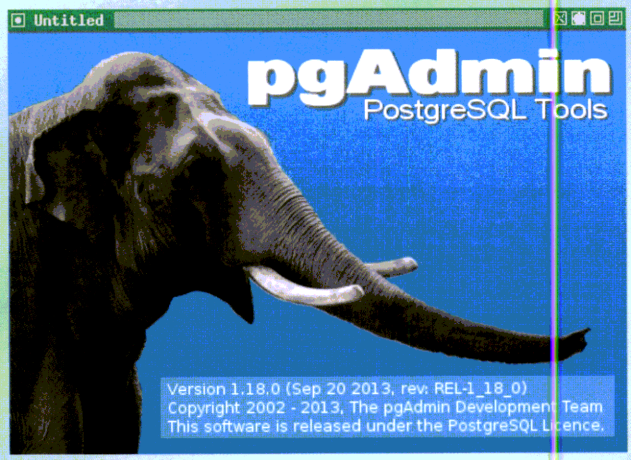
Man kan godt bruge et GUI program til at se eller rette på postgres databaser mv.og det er klart nok behageligt, at man får kommandoer serveret i en menu; men for nogle mennesker fungerer det ikke, fordi der er så mange detaljer, der er virkelig mange indbyggede ting i postgres, og derfor er det egentlig nemmere at komme igang ved at bruge simpelt SQL fra kommandolinien.

Pgadmin3 er kommet langt; 3-tallet viser, at der er sket en del udvikling og branches. Det vil vi ikke komme nærmere ind på her. Udpakning og kompilering går heldigvis nemt, efter samme mønster som postgres. Som root:

```
cd /usr/local/src/dbase
tar -xkzf /distfiles/pgadmin-1.18.0.tar.gz
cd pgadmin3-1.18.0
./configure && make && make install
```

Der vil nok mangle et par "-devel" pakker, headerfiler og support libraries, som man må finde (ligesom for postgres) og installere. Det behøver ikke at være fra source, med mindre man kører på en megen gammel installation.

Derefter kan man starte pgadmin3, men bemærk at man faktisk ikke får adgang til postgres databasen! For at få adgang til den skal man lave om på adgang til postgres-access kontrol, for pgadmin forbinder sig via TCP/IP.



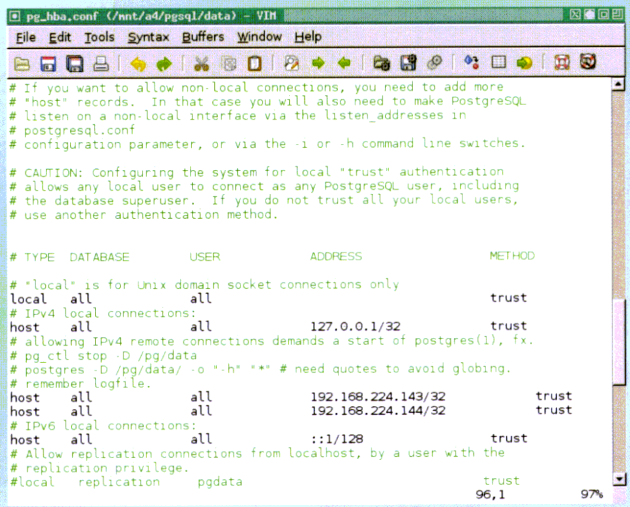
Access kontrol og konfigurationsfil

Den nødvendige linie i /usr/local/pgsql/data/pg_hba.conf

```
#Type    base user    address                 method
host     all  all     127.0.0.1/32            trust
```

Som det ses på billedet af konfigurationsfilen er der mange kommentarer. Det er simpelthen forbilledligt; de fleste at de kolleger og kursister, jeg har mødt, med lidt erfaring, nemt at rette den til, fanger hurtigt syntaxen i sådan en fil, og retter

eksemplerne til, så de passer. Læg mærke til, at eksemplet viser



en host pr. linie, men netmasken antyder, at man kan godt have netværk eller grupper på nettet som klienter.

Man behøver med andre ord ikke at skrive mere end højst nødvendigt.

Til en begyndelse er det nok at åbne for localhost, så vil pgadmin kunne forbinde sig til backend.

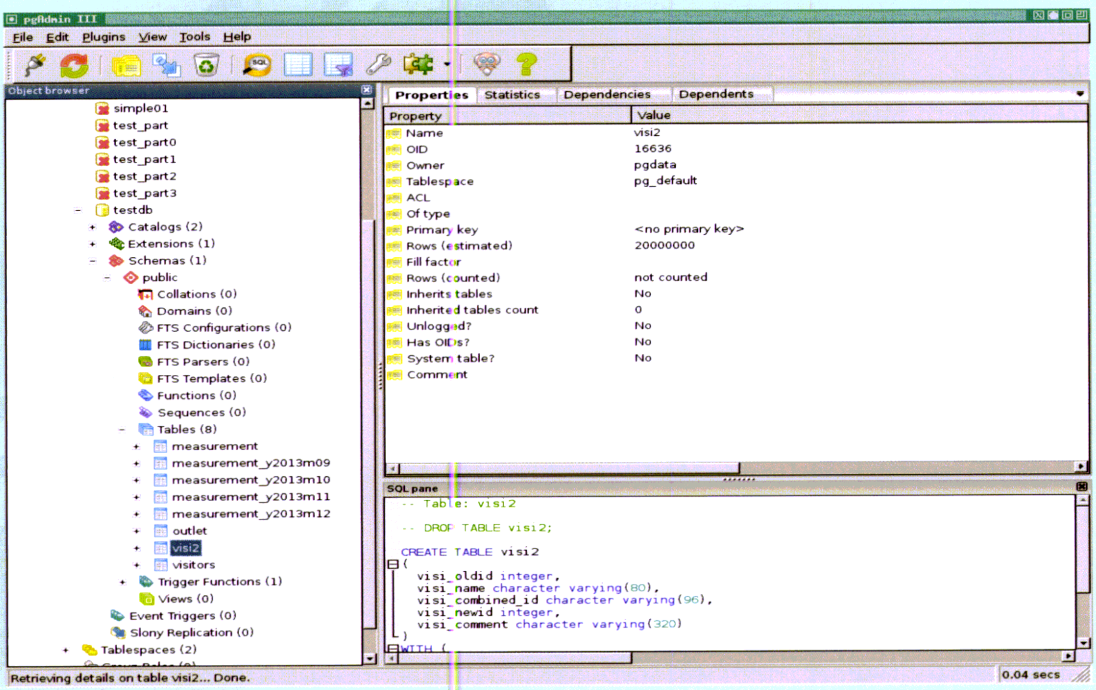
Men det er ikke nok - det står faktisk øverst i filen pg_hba.conf at man skal huske at rette backend konfigurationsfilen postgresql.conf, så den lytter på TCP/IP:

```
# what IP address(es) to listen on;
listen_addresses = 'localhost, 192.168.224.144'
```

I eksemplet er tilføjet en ekstern adresse ***(dog kun på lokalnettet beskyttet af en firewall!)***

Nu vil pgadmin forbinde sig, måske advare om at postgres brugeren ikke har noget password. Det bør man selvfølgelig oprette, og indtaste i pgadmin3 hver gang man opretter forbindelse, med mindre man befinder sig bag en låst dør.

Som det ses af skærmdumpet af pgadmin3 har redaktionen oprettet et kartotek med 20 millioner records. Det tager sin tid, selvfølgelig, men giver en god ide om, hvor effektiv softwaren er. På en almindelig workstation med AMD Athlon(tm) II X2 250 Processor 2700 MHz og 4 GB RAM kom vi op på ca. 1100



inserts pr. sekund i en uindexeret tabel, og 1500 inserts i sekundet med en base på 6 mio records på en lidt hurtigere maskine. De fleste maskiner med moderne hardware vil kunne opnå de resultater; omsat til transaktioner bliver det - igen, afhængig af hvad og hvor meget, nok til at kunne betjene ca. 50 samtidige brugere i sekundet. Det er fint for et lille system, men for store internationale web-services, fx. et elselskab med 10 mio kunder, er det selvfølgelig ikke nok, og det er derfor at forfatteren af nærværende artikel planlagde en række forsøg med partitionering og load balancing på de to maskiner, som redaktionen har til rådighed.

I kommandocentralen side 19 vises et script til at danne testdata.

En af de kendte metoder til at sprede belastningen ved databaseforespørgsler kaldes horisontal partitionering. Den består i at dele indgående opdateringer på en nøgle; det kan være den eneste nøgle, som der spørges på, og i så fald er metoden perfekt. I modsat fald får man et problem, idet forespørgsler skal sendes til alle maskiner med backends, samles sammen og leveres til applikationen. Undertegnede havde den fornøjelse at være i nærheden af den første store data-mining database i Danmark, en DB2 på AIX-systemer med disk-tower på ialt 4,000 MB (4 Tb) hos TDC, som i 1997 kunne sluge ca. 10 millioner records på 20 minutter. Det er pænt hurtigt, ca. 8000 i sekundet; men i modsætning til vores test her var det ikke inserts, men hele samlede multi-operationer, en proces, som kørte for sig selv. I Posgresql og andre systemer kaldes det COPY eller bulk copy. Det er ikke en del af SQL standarden.

Bruger man copy til at "oplade" 6 mio records til databasen, så tager operationen 32 sekunder, d.v.s. der indlæses 187500 records i sekundet på redaktionens langsomste workstation.

Bemærk at lange kommandolinierne fortsættes med backslash og nylinie. Der må ikke være space efter backslash!

```
$ time psql -q -U pgdata testdb\
 -f simdata/visitors-restore.txt
   32.72s real     3.18s user     0.58s system
$ wc /a4/simdata/visitors-restore.txt
  6000032  24000079 222652325 simdata/
                          visitors-restore.txt
$ psql -U pgdata testdb
psql (9.3rc1)
Type "help" for help.

testdb=# select count(*) from visitors;
  count
---------
 5999999
(1 row)

testdb=#
```

Belastningsprogram pgbench

Der følger en del tredieparts programmer med postgres, og et af dem, pgbench, giver en ide om egenskaberne ved forskellige typer belastning.

Men pgbench installeres ikke ved minimal build. Igen gælder det, at test-kørselen ikke kan foretages som administrator. Hvis source ejes af root, køres først chown:

```
chown -R postgres:postgres postgresql-9.3/
cd /.../postgresql-9.3/contrib/pgbench
make
make test
su -c "make install"
```

To kørsler med pgbench

Pgbench opretter sine egne test-databaser:

```
pgbench -i pgbench
```

Eller hvis man vil have 10 mio records:

```
$ pgbench -i -s 100 pgbench
```

Derefter kan man køre belastning af forskellig slags. Default (uden options) danner programmet en stpr tabel med 100000 records og nogle 3 små tabeller, og kører 1 klient, som kører 10 queries:

```
$ pgbench pgbench
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 100
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10
number of transactions actually processed: 10/10
tps = 156.184110 (incl connections establishing)
tps = 172.461369 (excl connections establishing)
$ time pgbench -c 50 -t 200
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 100
query mode: simple
number of clients: 50
number of threads: 1
number of transactions per client: 200
number of transactions actually processed:
10000/10000
tps = 248.245235 (incl connections establishing)
tps = 249.207986 (excl connections establishing)
   40.31s real     0.67s user     0.69s system
$
```

Et tal på mere end 100 T/sec var vanskeligt at opnå tidligere.

Man kan sætte ***time*** foran kommandoer (som i anden kørsel ovenfor) for at se, hvor lang tid kørselen har varet (+ system og user-tid).

PL/Proxy database partitionering

Database partitionering (opdeling af data-tabeller i flere, mindre tabeller) er en måde at fordele belastningen ved forespørgler på én tabel (en database) ud på flere maskiner.

PLProxy arbejder sammen med Management of External Data, (SQL/MED, en ISO standard) som sørger for access-kontrol og data-repræsentation. Der findes alternative måder at tilgå eksterne databaser, først og fremmest *dblink*.

PL/Proxy - Hvad er det?

PLProxy er en udvidelse af PL/pgSQL, som svarer til Oracles PL/SQL, Procedural Language (for SQL database access). PLProxy kan køre på en, nogle eller alle medlemmer af et cluster, og kører parallelt hvis query retter sig til flere partitioner. Kommunikationen er optimeret. PLProxy kan arbejde sammen med PgBouncer for at opnå connection pooling (én forbindelse bruges til mange forespørgsler, optimering af nettrafik).

På minussiden står, at hver query bliver autocommit, og at rollback (annullering midt i en sekvens af opdateringer) derfor ikke kan ske.

Et spørgsmål popper uvægerligt op: hvad skal vi med PLProxy, når vi har SQL/MED, dblink, mv.? Svaret, sådan som udviklerne giver det, er, at plproxy giver mulighed for store udvidelser, *sharding,* af databaseproblemer, og ikke ændrer på postgres' programstruktur. Udviklerne nævner andre mulige anvendelser af PLProxy, bl.a.simpel read-only load balancing.

PL/Proxy - installation

Skal installeres fra source, som hentes i pgfoundry. Samme procedure som i foregående eksempel. Kør en test:

```
make installcheck
```

Scriptet usr/local/pgsql/share/extension/plproxy--2.5.0.sql skal køres for hver ny database (hvis man altså vil bruge plproxy).

Der er kun fire statements i plproxy:

CONNECT, specificerer forbindelse til anden db.

CLUSTER, hvilket cluster skal der køres på?

RUN ON, RUN ON ALL fx. kører på hele cluster

SELECT, Kan bruges til at specificere en query - hvis ikke der specificeres noget, vil plproxy kalde en funktion som svarer til eget funktionsnavn.

Der er nogle flere nøgleord, keywords, som er underafdelinger af de fire overordnede. I eksemplet her bruges de tre sidste.

Eksempel på horisontal partionering

Vi vil opdele en database i partitioner (her kun 2) som ligger på hver sin server. Opdelingen foretages med en hash på det felt, som skal være søgefelt, og det er selvfølgelig en begrænsning. I den nye postgres er forbindelse til externe databaser forbedret, og den benytter vi sammen med CLUSTER til at skabe en forbindelse - obs: ordet cluster bruges i en anden betydning her.

Alene kan plproxy ikke løse opgaven, det første vi gør, er at oprette et cluster via SQL/MED funktionaliteten. Clusteret består af en masterdatabase, proxy'en, hvorigennem alle forespørgsler kanaliseres ud på andre maskiner (ideelt fx. 16 maskiner, her kun den lokale maskine plus nabo-maskinen).

```
-- Dette er en kommentar.
create database simple;
create database simple01;
\c simple01
create table usemail (
    username text,
    email text
);

-- kan også gøres ved at logge ind på remote
-- maskinen, selvfølelig:
-- connect to db as user on host:
\c postgres postgres nabo
create database simple02;
\c simple02 postgres nabo
CREATE TABLE usemail (
    username text,
    email text
);
\c simple;
-- Fem ting til at definere cluster:
CREATE SERVER mailcluster FOREIGN DATA WRAPPER
plproxy
OPTIONS (connection_lifetime '1800',
p0 'dbname=simple01 host=127.0.0.1',
p1 'dbname=simple02 host=192.168.224.143' );

CREATE USER MAPPING FOR public SERVER
mailcluster;

CREATE USER MAPPING FOR pgdata SERVER
mailcluster OPTIONS (
user 'locodon', password 'simsala');

GRANT USAGE ON FOREIGN SERVER mailcluster TO
pgdata;
```

Bemærk at den lokale bruger pgdata (superbruger) mapper til en anden bruger med password på den remote database. plproxy og SQL/MED accepterer ikke at udføre en connection til en passwordløs remote superbruger (af sikkerhedshensyn). Vi går ikke i dybden med sikkerhedsspørgsmål.

Cluster og bruger og access er nu defineret. Derefter skrives et minimal sæt af funktioner for at demonstrere, hvad man kan.

Der skal være lokale "insert" funktioner, dem opretter vi først:

```
\c simple01
CREATE OR REPLACE FUNCTION
insert_user(i_username text, i_emailaddress
text)
RETURNS integer AS $$
       INSERT INTO usemail (username, email)
VALUES ($1,$2);
       SELECT 1;
$$ LANGUAGE SQL;
\c simple02 pgdata nabo
CREATE OR REPLACE FUNCTION
insert_user(i_username text, i_emailaddress
text)
RETURNS integer AS $$
       INSERT INTO usemail (username, email)
VALUES ($1,$2);
       SELECT 1;
$$ LANGUAGE SQL;

\c simple

CREATE OR REPLACE FUNCTION
insert_user(i_username text, i_emailaddress
text)
RETURNS integer AS $$
    CLUSTER 'mailcluster';
    RUN ON hashtext(i_username);
$$ LANGUAGE plproxy;

CREATE OR REPLACE FUNCTION
get_usemail(i_username text)
RETURNS SETOF text AS $$
    CLUSTER 'mailcluster';
    RUN ON hashtext(i_username) ;
    SELECT email FROM usemail WHERE username =
i_username;
$$ LANGUAGE plproxy;
```

Det er linien RUN ON hashtext(i_username) som beregner hvilken maskine den enkelte key skal sendes til. Egentlig hashtext('xyz') binary AND 1, fordi RUN ON reducerer det høje tilsendte tal med bitmaske. Det er så også grunden til, at der skal være 2^n subpartitioner (et antal potens af 2) - men de behøver ikke at ligge på hver sin maskine.

Nu kan vi begynde at bruge clusterfunktionerne:

```
SELECT insert_user('Alf','alf@gmail.com');
SELECT insert_user('Bent', 'btyde@gmail.com');
SELECT insert_user('Chris','cjensen@mail.dk');

SELECT get_usemail('Alf');
SELECT get_usemail('Bent');
SELECT get_usemail('Chris');
```

Der er selvfølgelig mange faldgruber. Nogle steder er dokumentationen uhyre kortfattet eller mangelfuld. Et eksempel: i tutorial til plproxy manglede *foreign* i sætningen grant usage on foreign server mailcluster to pgdata; og det giver en fejlmeddelelse og giver derfor ikke den nødvendige tilladelse (grant = tilladelse).

Man skal også huske, at dette setup bevirker, at man ikke kan indsætte noget i databaserne lokalt, alt skal gå gennem *hashtext()* fordi den beregner cluster-nummer ud fra den tilsendte key. Men data fra de lokale tabeller kan godt tilgås via SQL, og derved vil man kunne sikre og evt. samle data sammen igen for måske at splitte op på en anden måde end ved hashtext(username).

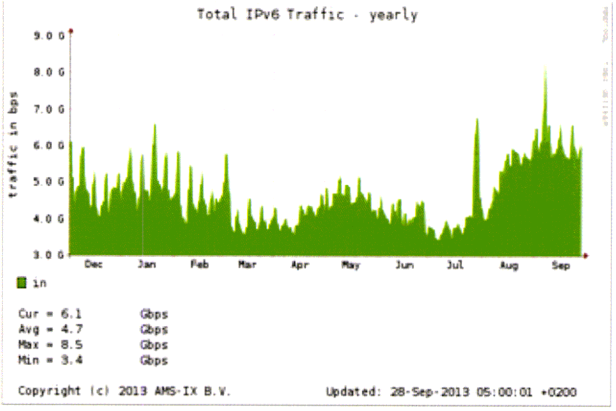
Opsummering

Det ses af eksemplet, at partitionering kræver meget programmering og kræver at alt SQL pakkes ind i PL/pgSQL. Det skal understreges at det er én af mange måder at sprede belastning.

I næste nummer kommer et eksempel på en web-applikation som foretager opslag i postgres.

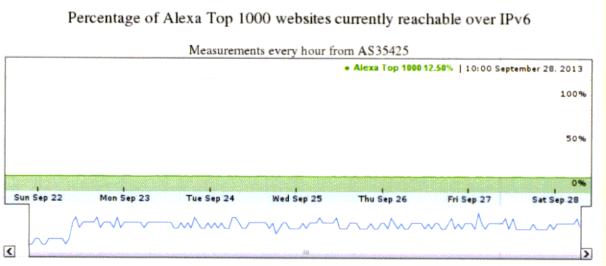
■

IPv6 status 1 år efter Launch Day



Trafikken er fortsat meget lille. Måske er det derfor, at vi, der bruger IPv6, oplever gode, hurtige forbindelser ☺

Percentage of Alexa Top 1000 websites currently reachable over IPv6



Her ses at ca.12% af top-website kan tilgås via IPv6. Der er små udsving op og ned - driftsforstyrrelser, som er forstørret på den nederste, blå kurve.

■

Kommandocentralen

Generer data til simulering

Et af de mest fleksible værktøjer er Gawk. Shell programmering består ofte i små moduler, som er tænkt som en option til andre programmer.

For hands on med gawk skal man ofte bruge nogle kolonner med tal - vær opmærksom på, at du selv skal sørge for, at der er en gawk i /bin (noge distroer lægger den i /usr/bin):

```
#!/bin/gawk -f
BEGIN {
  #srand();# for rand() ny hver gang
  x=0;
  while (++x < 200) {
     printf("%6.4f %6.4f %6.4f \n",
          rand(),rand(),rand());
  }
  exit(0);
}
```

For at holde kildetekstens bredde nede kan man dele argumenter til en funktion efter hver parameter, helst efter kommaet, som her efter format-streng og inden rand(). Man må ikke dele en streng parameter.

Gawk-sproget er ikke helt som C, fx. er semikolon og linieskift er ækvivalente, med andre ord, man behøver ikke at sætte semikolon for enden af en linie. Variable bliver skabt ved at blive nævnt, og kommentarer er i shell-syntax, men ellers er det meget som C.

Ovenstående vil danne samme tal hver gang den køres. Hvis man ønsker genereret en anden sekvens af tal, er man nødt til at seede rand() funktionen, srand(), ved at fjerne hash (havelåge, kommentar-) tegnet foran.

For at generere random data til insert i en database kan vi bruge samme teknik:

```
#!/bin/gawk -f
BEGIN {
  srand();
  x=0; big=1000000;
  while (++x < 10) {
     printf(\
"insert into visitors values (%d,'%s',%d);\n",
          big*rand(),rand(),big*rand());
  }
  exit(123);
}
```

Hvis man har brug for at dele funktionslinien, kan det gøres sådan (med backslash+linieskift efter første parentes).

Der er sat **single-quotes** omkring string parametrene. Det er SQL måden at angive en **streng (text-værdi).** Bemærk at man godt kan give et tal (her: retur værdien af rand()) til en %s - talværdien bliver automatisk omdannet til streng.

Men det ser jo ikke kønt ud at bruge tal i stedet for navne. Der er flere forskellige måder at danne random strings, som indeholder noget der ligner tekst, og det er faktisk meget rart til en demo-applikation, at der ikke kun står *qwerty* og *hjkhjkhjk*. Vi kan fx. indexere os ind i et array af strenge, som udgør dele af almindelige navne:

```
[...] # del af awk program
# hvor der dannes pseudonavne:
BEGIN{
  init_nameparts();++lbnr;
  name = pf[int(rand()*1000000)%64]\
     ps[int(rand()*1000000)%128]
# hvis man vil debugge:
# print name; exit(123);
  printf(\
"INSERT INTO visitors VALUES (%d, '%s', %08d);\n",
  rand()*10000000, name, lbnr);
}
function init_nameparts() {
    pf[0] = "Ander";
    pf[1] = "Atte";
    pf[2] = "Beta";
[...]
    pf[63] = "Ilse";
    pf[64] = "Iste";

    ps[0] = "son";
    ps[1] = "sen";
    ps[2] = "berg";
[...]
    ps[127] = "bach";
    ps[128] = "buch";
```

Eksemplet er uddrag af kode. Af hensyn til overskuelighed er streng-dele ikke færdiggjort. Ved fx. 64 x 128 delelementer fås ialt 8192 forskelllige navne-strenge.

Der mangler ligeledes loop for generering af millionvis af data - det overlades til læseren at skrive programmet færdigt. En færdig version kan dog hentes på http://tiny.cc/q9b43w

Database - tabel til denne øvelse kan laves således:

```
create table visitors(
   visi_id int,
   visi_name text,
   visi_oldid int);
```

Til big-table performance experimenter i PostgreSQL artiklen var der flere felter, - man kan tilføje felter efter behag.

■

SUPERUSERS

Vores passion for programmering kommer dig til gode, når du er på kursus hos os

Intro til objektorientering og programmering

NR.	TITEL	DAGE
SU-0199	Introduktion til Programmering	2
SU-0202	Objektorienteret Grundkursus for C++/C#/Obj-C og Java	2

Design og udvikling

NR.	TITEL	DAGE
SU-0205	UML (Unified Modelling Language)	3
SU-0255	Design Patterns	2

C / C++ / Objective C / C#

NR.	TITEL	DAGE
SU-0200	C Programmering Grundkursus	4
SU-0201	C Programmering Videregående	3
SU-0203	C++ Programmering Grundkursus	5
SU-0204	C++ Programmering Videregående	5
SU-0207	C# / VB.NET Programmering Intro	3
SU-0208	Obj-C Programmering (inkl. Mac Mini)	5
MS-0483	(20483) Programming in C#	5

Java

NR.	TITEL	DAGE
SU-0210	Java Programmering Grundkursus	5
SU-0211	Java Programmering Videregående	5
SU-0212	Java 7 og Java 8 upgrade	2
SU-0218	Java og XML Development	4

Perl / PHP

NR.	TITEL	DAGE
SU-0220	Perl Programmering Grundkursus	4
SU-0221	Perl Programmering Videregående	4
SU-0230	PHP Programmering Grundkursus	3

SQL

NR.	TITEL	DAGE
SU-0239	Database Design Grundkursus	2
SU-0240	SQL Programmering Grundkursus	3
SU-0241	SQL Programmering Videregående	3
SU-0242	MS Transact-SQL (T-SQL Progr.)	3

Apple iOS

NR.	TITEL	DAGE
SU-0208	Objective C Programmering	5
AP-0901	iPhone/iPad iOS Programmering Grundkursus (inkl. iPad Mini)	2
AP-0902	iPhone/iPad iOS Programmering Workshop	3
AP-0904	iPhone/iPad iOS Programmering af brugergrænseflade	3
AP-0905	iPhone/iPad Programmering Videregående (inkl. iPhone 5)	3

Android

NR.	TITEL	DAGE
LX-0901	Android Programmering Grundkursus	2
SU-0210	Java Programmering Grundkursus	5
LX-0902	Android Programmering Workshop	3
LX-0904	Android Progr. Brugergrænseflade	3
LX-0905	Android Progr. Videregående	3

Windows Phone 8

NR.	TITEL	DAGE
MS-0901	Windows Phone 8 Programmering Grundkursus (inkl. Windows Phone 8)	2
MS-0483	(20483) Programming in C#	5
MS-0902	Windows Phone 8 Programmering Workshop	3
MS-0904	Windows Phone 8 Programmering af brugergrænseflade	3
MS-0905	Windows Phone 8 Programmering Videregående	3

Web Programmering og Web Apps

NR.	TITEL	DAGE
SU-0901	Web App Programmering Grundkursus (inkl. Mobile Device)	2
SU-0902	Web App Programmering Workshop	3
SU-0904	Web App Programmering af brugergrænseflader	3
SU-0905	Web App Programmering Videregående	3
SU-0093	jQuery – Det samlede client web-udviklingsforløb	3
SU-0095	Node.js - Det samlede server web-udviklingsforløb	3

Se kursusbeskrivelser og datoer på superusers.dk/programmering.htm

Tilmeld dig via telefon 48 28 07 06 eller mail super@superusers.dk

Karlebogaard: Karlebovej 91 • 3400 Hillerød & **Kampehøjgaard**: Krajbjergvej 3, Vorre • 8541 Skødstrup
Tlf.: 48 28 07 06 • **Mail**: super@superusers.dk • **Web**: www.superusers.dk • **Alle priser er ekskl. moms**