DKuug nr. 169 December 2012

NYT

Dansk Forum for Åbne Systemer

DKUUG - Unix Brugere (Linux/BSD) system administratorer
Mødested for IT-specialister og IT-interesserede.

DKUUG

Fra redaktionen

Identitetstyveri

Fremtidsforsker Liselotte Lyngsøe blev en dag pludselig ringet op af sit forsikringsselskab med besked om at hun ville stige i præmie fordi hun var flyttet til Valby. Men det kunne jo ikke passe, for hun var ikke flyttet!

Liselotte Lyngsøe skyndte sig at ringe til sin kommune, hvor hun fik beskeden "Desværre Liselotte, vi kan ikke hjælpe dig, du er fraflyttet."

"Hvad er jeg? I kan jo se at jeg ikke kan have børnene boende alene på min adresse i Gentofte!"

Liselotte Lyngsøe er mor til 4 børn, og dermed bliver hun afkrævet sit CPR-nummer alle mulige steder, hos læger, tandlæger, skoler, børnehaver. "På den måde er min profil meget sårbar. Jeg er nødt til at være tilgængelig mange steder."

Som fremtidsforsker er hun godt klar over, at vi bliver mere og mere usikre, men hun er også klar over, at vi ikke kan afvise de nye måder at administrere.

Ved henvendelse til Borgerservice er det ikke nok at flytte tilbage eller annullere flytningen, for der er et anciennitetsbegreb, som går fløjten, hvis man har boet udenfor kommunen. Det tager én dag. Så skulle den sag være klaret - men:

"14 dage senere vælter det ind ad brevsprækken med tyvekoster: Billetter til Roskildefestivalen, og, mere skræmmende, forskellige kontokort til forskellige banker. Der er åbnet 8 mobiltelefoner i mit navn. Jeg henvender mig til politiet, som må kunne fange tyvene ved den gamle adresse, for de ved endnu ikke at jeg har rettet adressen! Men politiet har slet slet ikke ressourcer til det."

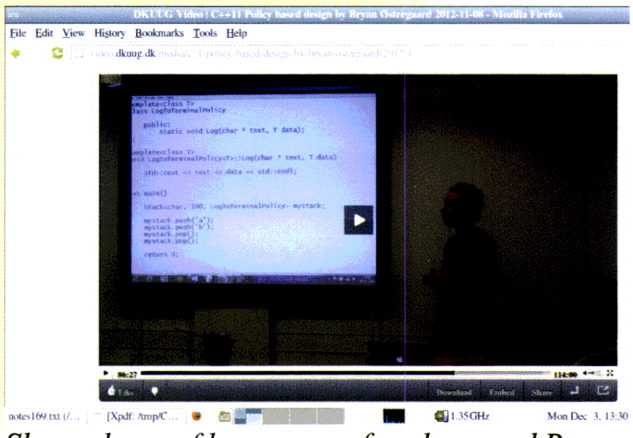
Og Liselotte måtte også ringe rundt og fortælle at det var en politisag og at hun ikke havde købt/oprettet etc. og hver gang bliver hun skældt ud. Og hun finder ud af at DK i den henseende er kendt som et slaraffenland for identitetstyve, fordi der ikke er ordentlige forebyggende ordninger.

Er det en historie, som vil gentage sig oftere? Har vi her i vores dejlige Danmark indført IT med hovedet under armen?

DKUUG er repræsenteret i RFSITS - Rådet for større IT-Sikkerhed, som nu er ved at organiseretet nyt udvalg, der skal være vejledende for myndighederne i sikkerhedsspørgsmål. Nu er det op til brugerne - os - at lægge pres på myndighederne så de lytter til det nye udvalg og ikke bare bruger det som gummistempel. Vores mand i RFSITS, Michael Lind Mortensen, er særdeles kompetent ligesom mange af vores nye medlemmer, og vi bør være med til at påpege og afhjælpe de skandaløse IT-sikkerhedsmæssige mangler her i landet.

Foredrag i DKUUG

Vedrørende sikkerhed skal det nævnes, at Bryan Østergaard holder foredrag om kryptering - eller rettere brydning af samme - d. 18. december, se foredragslisten side 3 - Det giver forhåbentlig anledning til en rigtig god julestemning!



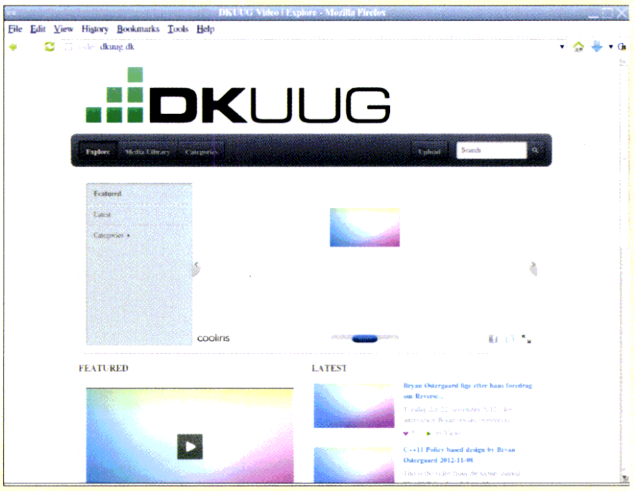
Skærmdump af browser m. foredrag med Bryan Østergaard, C++11 Policy Based Design

Bryan Østergaard har holdt en serie foredrag om C++11 - en videreudvikling af C++. (C++ fra år 2011; den gamle standard var C++ fra 1998 kaldet C++98).

Bryans emneliste omfattede "C++11 Policy Based Design", som er en konvention baseret på templates. Der er ikke noget i vejen for at bruge policy based design i andre sprog, fx. Javascript, men
C++11 har mange nyskabelser, som hjælper programmøren med adskillelse af algoritme, datatype og datarepræsentation.

Hvad er det nye ved C++11? Hvorfor mere, hvorfor nyt? Gik du glip af foredragene? Så kan du nu se dem på video.dkuug.dk.

Websitet er en hurtig opsætning af MovieMasher, som Jon Bendtsen har foretaget - og det ser tiltalende ud!



Mediacore er et meget nyttigt Open Source projekt - her ses hovedskærmen og lidt af indholdsfortegnelsen nedenunder

Artiklerne i dette nummer er forhåbentlig dejligt nørdede inspirerende artikler som kan bruges som guide til eksperimenter i juleferien. Mikkel Mikjær har skrevet om Galera, et high availability multi-master cluster for MySQL - med InnoDB, og artikelen giver en god fornemmelse af installation og checkmulighed.

Hjerre F. Hviid har skrevet om pointere i C og i Java, noget, som næsten altid vækker gru hos begynder-programmører, især hvis man kommer fra et sprog, som pakker computerens funktionalitet ind i et utal af abstraktioner. Men der er pointere i alle sprog, og vi får syn for sagn i Hjerres artikel!

Ole Tanges artikel om Parallel er et godt bud til effektivisering af scripts - det er et meget intelligent multitasking xargs-lignende værktøj.

DK-uug NYT har i nogen tid efterlyst et nyt, mere sigende navn. Der er ikke længee så mange, der ved hvad Unix er, og lyden af navnet inspirerer lidt til at gøre grin med foreningen. På den anden side er det et kendt *brand (kendetegn, varemærke)* i miljøet, og det må vi ikke miste. Vi har ikke tænkt os at ændre noget radikalt på en gang - men vil gøre det gradvist, ligesom dengang Landmandsbanken gik over til navnet Danske Bank, først med et hvidt "L" afgrænset af tre røde felter; - der manglede blot én rød firkant for at det var Dannebrog. Efter nogen tid blev dette logo ændret til et "rigtigt" flag. Efter sammenslutningen med Handelsbanken (som var den store i den sammenhæng) fik man så et helt andet logo, men det er en anden historie.

Alle forslag til *branding* af bladet er velkomne. Foreløbig nøjes vi med i daglig omtale at kalde det DNyt.

God læselyst

Donald Axel

DKUUG-Nyt er medlemsblad for DKUUG, foreningen for Åbne Systemer og Internet

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

Forsidefoto: Donald Axel
Design og layout:
DKUUGs PR-gruppe
Annoncer:
pr@dkuug.dk
Tryk:
Digital-trykkeriet i Aarhus
Oplag:
500 eksemplarer

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

Vores møder og **foredrag** holdes - med mindre andet utrykkeligt 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 i den kommende sæson:

OSD 2013 lørdag-søndag d. 9-10 marts 2013. Der vil muligvis blive kurser dagen før.

thecamp.dk har lagt en lang række video'er på **video.thecamp.dk**

Foredrag i Symbion:

Tirsdag d. 18. december kl.19: Hvordan brydes kryptografi?

Et sjovt foredrag der giver et basalt indblik i nogle af de metoder der bruges til at bryde krypteret data. Der bliver vist praktiske måder at bryde mange simple krypteringsalgoritmer men der vil også blive gennemgået nogle svagheder ved moderne algorithmer som for eksempel RSA.

Onsdag d. 9. januar 2013 kl. 18: Workshop om blad-artikler.

2-3 timers arbejde ud fra emnet "Troværdighed". Har D(Kuug)Nyt et godt omdømme? kan det bruges til fremme af ideen om at åbne systemer er en nødvendighed? At åbne systemer, baseret på standardiserede specifikationer af programmeringsinterfaces **ikke kun er en sag for fagfolk** men er et politisk anliggende? Kan vi forklare tingene godt nok? I den forbindelse skal det tilføjes, at de næste numre af D(Kuug)Nyt bliver brugt til promotion på udvalgte institutioner.

Kom og få dine kompetencer plejet - hold et foredrag om det, der interesserer dig (og os).

Vores kontor i Symbion er i orden nu; vi nu kan have gruppe-arbejde og 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.

Deadline for DNyt nr. 170: Søndag d. 10. februar 2013

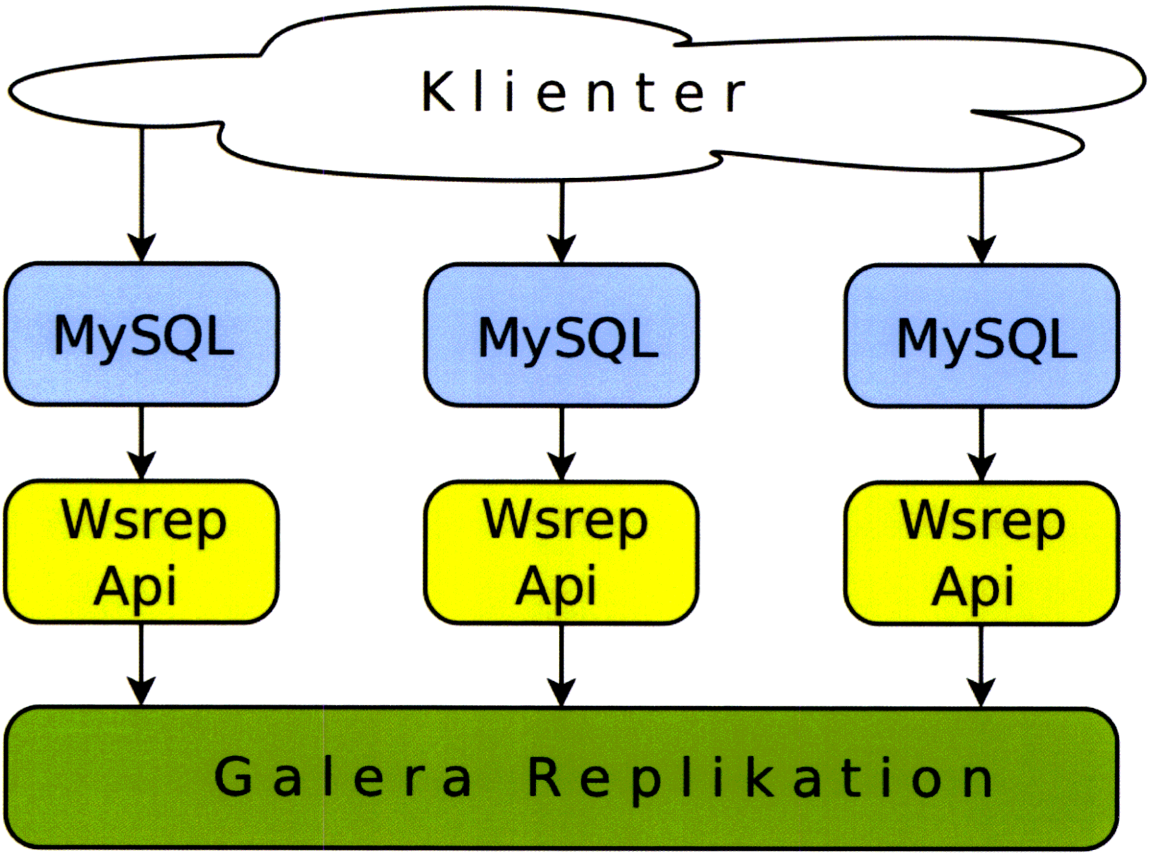
Ny indretning i Symbions indgangshal giver mulighed for hyggesnak
*Forsiden er Indgangshallen i Symbion hvor DKUUG - **Dansk Forum for Åbne Systemer** - har kontor - med juletræ. Det anes at der er en sofagruppe i forgrunden. Der er siden kommet lænestole, skillevægge og kaffemaskine til, og man er velkommen til en hyggesnak før foredrag eller workshop møder.*

High Availability, MySQL Galera

Galera er et synkront multi-master cluster for MySQL/InnoDB databaser. Eller med andre ord, et af de meste effektive opensource værktøjer til at lave redundante applikationer.

Et typisk setup består af 3 servere, og din applikation kan tilgå en vilkårlig server på et vilkårligt tidspunkt, og hvis en af serverne falder ud kan en af de andre uden videre benytte en af de andre uden yderligere driftsforstyrrelser.

Af Mikkel Mikjær Christensen



Det betyder i modsætning til klassisk MySQL replikation at du ikke behøver modificere din applikation til f.eks. at lave read-write-split [1], write-operationer udføres synkront på alle serverne, det betyder at den langsomste server sætter grænsen for databasens ydelse ifbm. skrive operationer, mens læse operationer kører med hastigheden på den individuelle server den nu afvikles fra.

Man skal være opmærksom på at ens schema skal konverteres til at køre ren InnoDB og man skal huske at have Primary Keys på alle tabeler, og man skal være forsigtig med at tilføje auto-increment på eksisterende tabeller, og generelt skal man bruge det til drift ikke til udvikling, da der kan opstå en række problemer ved at ændre på databaser i drift med data.

Galera er klar til produktion, og det er en udmærket drop-in replacement for MySQL, men man er nødt til selv at køre intensiv test på systemet og du er, som ved alle andre opensource projekter, nødt til at danne dig dine egne erfaringer.

Jeg har sat 3 virtuelle servere op, vps1, vps2 og vps3 som alle kører Debian, og så kan vi jo passende lægge ud med at installere en håndfuld dependencies fra Debians repository:

```
root@vps1:~# apt-get install libdbi-perl\
libdbd-mysql-perl mysql-client-5.1 rsync\
libmysqlclient16 mysql-common \
libplrpc-perl libnet-daemon-perl vim \
libaio1 psmisc
```

Galera består dels af en patched MySQL og så af Galera selv, så vi henter de officielle Debian pakker:

```
root@vps1:~# wget \
http://launchpad.net/codership-\
mysql/5.5/5.5.23-23.6/+download/mysql-\
server-wsrep-5.5.23-23.6-i386.deb

root@vps1:~# wget \
https://launchpad.net/galera/2.x/\
23.2.1/+download/galera-23.2.1-i386.deb
```

og installerer dem:

```
root@vps1:~# dpkg -i galera-23.2.1-
i386.deb mysql-server-wsrep-5.5.23-23.6-
i386.deb
```

1) Read-write-split betyder at din applikation benytter én eller flere specifikke database servere til læse-operationer og én anden specifik database server til skrive operationer.

Herefter er selve galera og mysql, med wsrep patchen, som den kaldes, klar til brug. Inden vi joiner clusteret skal vi dog have opsat mysql, dette gøres nemmest med mysql_secure_installation:

```
root@vps1:~# /etc/init.d/mysql start
Starting MySQL database server: mysqld.
root@vps1:~# mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MySQL
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!


In order to log into MySQL to secure it, we'll need the current
password for the root user.  If you've just installed MySQL, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.

Enter current password for root (enter for none): <enter>
OK, successfully used password, moving on...

Setting the root password ensures that nobody can log into the MySQL
root user without the proper authorisation.

Set root password? [Y/n] y
New password: <password>
Re-enter new password: <password gentaget>
Password updated successfully!
Reloading privilege tables..
 ... Success!


By default, a MySQL installation has an anonymous user, allowing anyone
to log into MySQL without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] y
 ... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] y
 ... Success!

By default, MySQL comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] y
 - Dropping test database...
 ... Success!
 - Removing privileges on test database...
 ... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] y
 ... Success!

Cleaning up...



All done!  If you've completed all of the above steps, your MySQL
installation should now be secure.

Thanks for using MySQL!
```

Dernæst skal vi have MySQL til at acceptere eksterne forbindelser, dette gøres ved at forbinde til den:

```
root@vps1:~# mysql -p
Enter password: <password>
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.5.23 Source distribution, wsrep_23.6.r3755

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SET wsrep_on = OFF; grant all on *.* to 'root'@'%' identified by '<password>';
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> quit
Bye
```

og så skal mysql lukkes ned så vi kan sætte Galera op:

```
root@vps1:~# /etc/init.d/mysql stop
Stopping MySQL database server: mysqld
Killing MySQL database server by signal:
mysqld . . . ..
```

Gentag nu ovenstående for alle dine noder, og så er du klar til at sætte selve clusteret op. Det gøres ved at redigere filen /etc/mysql/conf.d/wsrep.cnf, på alle noder, start med at rette:

```
# Full path to wsrep provider library or 'none'
wsrep_provider=none
```

til:

```
# Full path to wsrep provider library or 'none'
wsrep_provider=/usr/lib/galera/libgalera_smm.so
```

Dette fortæller wsrep-patchen, hvilket modul der skal benyttes til replikeringen, i det her tilfælde er det Galera, som skal anvendes.

Dernæst skal du for vps1 ændre:

```
# Group communication system handle
#wsrep_cluster_address="dummy://"
```

til dette:

```
# Group communication system handle
wsrep_cluster_address="gcomm://"
```

og for de to andre til:

```
# Group communication system handle
wsrep_cluster_address="gcomm://vps1.eksempel.dk"
```

Dette skyldes at Galera ikke vil starte hvis ikke den kan finde resten af clusteret, derfor skal vi sætte adressen til gcomm:// og dermed fortælle Galera at den skal starte alligevel, og dermed initiere et nyt cluster.

Hvis dit primære netkort hedder eth0 behøver du ikke næste skridt, men for langt de fleste database installationer vil dette ikke være tilfældet, typisk vil man bruge bonding el. lign. for at lave redundant link mellem databaseserverne. Men i det her tilfælde bruger jeg virtuelle servere uden "eth0" derfor sætter jeg manuelt:

```
# Address on THIS node to receive SST at.
# DON'T SET IT TO DONOR ADDRESS!!!
# (SST method dependent. Defaults to the
# first IP of the first interface)
# wsrep_sst_receive_address=
```

til (den enkelte servers eget hostname / ip adresse)

```
# Address on THIS node to receive SST at.
# DON'T SET IT TO DONOR ADDRESS!!!
# (SST method dependent. Defaults to the
# first IP of the first interface)
wsrep_sst_receive_address=vps1.eksempel.dk
```

Til sidst retter du:

```
# State Snapshot Transfer method
wsrep_sst_method=mysqldump
```

til:

```
# State Snapshot Transfer method
wsrep_sst_method=rsync
```

SST står for State Snapshot Transfer og det handler basalt set om at sørge for at dine noder er enige om hvilken database de tager udgangspunkt i, den hurtigste måde at lave SST på er med rsync, desuden understøtter den installation jeg har guidet dig til at lave ikke andet, fordi Debian's repository kun indeholder mysql klienten i version 5.1 mens Galera's patchede udgave er version 5.5.

Så er du ved at være klar til start ...

Når du har udført ovenstående starter du MySQL på vps1:

```
root@vps1:~# /etc/init.d/mysql start
Starting MySQL database server: mysqld.
```

Og nu kan du teste at alting er som det skal være:

```
root@vps1:~# mysql -p
Enter password: <password>
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.57 wsrep_0.8.2.r3121

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql> show status like "wsrep_ready";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wsrep_ready   | ON    |
+---------------+-------+
1 row in set (0.00 sec)

mysql> show status like "wsrep_cluster_size";
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 1     |
+--------------------+-------+
1 row in set (0.01 sec)

mysql> show status like "wsrep_cluster_status";
+----------------------+---------+
| Variable_name        | Value   |
+----------------------+---------+
| wsrep_cluster_status | Primary |
+----------------------+---------+
1 row in set (0.00 sec)

mysql>
root@vps1:~# /etc/init.d/mysql start
Starting MySQL database server: mysqld.
```

Bemærk at wsrep_ready står til "on" og wsrep_cluster_status står til "Primary" mens wsrep_cluster_size står til "1" - det er et ret lille cluster vi har fået samlet os, så lad os gøre noget ved det:

```
root@vps2:~# /etc/init.d/mysql start
Starting MySQL database server:
mysqld ......
```

Det kan muligvis tage lidt lang tid for den at starte, men når den er færdig skulle du gerne kunne forbinde til den og teste:

```
root@vps2:~# mysql -p
Enter password: <password>
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.1.57 wsrep_0.8.2.r3121

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql> show status like "wsrep_cluster_size";
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 2     |
+--------------------+-------+
1 row in set (0.00 sec)

mysql>
```

Det ser jo lovende ud, lad os teste det lidt yderligere, først VPS1:

```
root@vps1:/# mysql -p<PASSWORD> -e 'show databases;'
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
+--------------------+
```

og på VPS2:

```
root@vps2:~# mysql -p<password> -e 'show databases;'
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
+--------------------+
```

Opret så en database på VPS1:

```
root@vps1:/# mysql -p<password> -e 'create database dkuug;
```

og se om den er oprettet på VPS2:

```
root@vps2:~# mysql -p<password> -e 'show databases;'
+--------------------+
| Database           |
+--------------------+
| information_schema |
| dkuug              |
| mysql              |
+--------------------+
```

Sørme ja ;-)

Nu mangler du bare at rette i /etc/mysql/conf.d/wsrep.cnf på VPS1:

```
# Group communication system handle
wsrep_cluster_address="gcomm://"
```

Det ændres til:

```
# Group communication system handle
wsrep_cluster_address="gcomm://vps2.eksempel.dk"
```

Det du her siger er, at hvis VPS1 mister forbindelsen til clusteret skal den ved genstart forsøge at connecte til den adresse igen, i modsat fald ville VPS1 have startet sit eget cluster, med de data den havde på disken, og dermed have forårsaget det som kaldes en ***split-brain,*** dvs to databaser der lever hver sit liv og selv tror at de har det eneste gyldige datasæt.

Derudover er det sådan, at når én node joiner clusteret vil en af de andre noder fungere som donor og lave en SST, og mens noden er donor, vil den ikke kunne svare på forespørgsler fra andre; af disse to grunde er det vigtigt, at du altid har mindst 3 noder i dit cluster. Derfor:

```
root@vps3:~# /etc/init.d/mysql start
Starting MySQL database server: mysqld.
```

Og:

```
root@vps1:~# mysql -p<password> \
 -e "show status like 'wsrep_cluster_size';"
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 3     |
+--------------------+-------+
```

Det er vigtigt at understrege at multi-master databaser og tilhørende applikationer ikke er noget man lærer at lave over-night, her hjælper jeg dig med at snuse til teknikkerne og giver en hurtig introduktion, og det kan ikke siges nok at du er nødt til at læse, eksperimentere, teste og generelt danne dig en masse erfaringer før du kaster dig ud i at lave kritisk produktion med det her.

Med 100GB data i en Galera replikeret database på moderne dual quadcore maskiner med 15.000 RPM Harddiske og 1000mbit netværk tager det cirka 15 minutter at lave en fuld SST, som med alle andre kritiske systemer skal du selvfølgelig være forberedt på hvad der sker når[2] det går galt og hvordan du opdager og ikke mindst håndterer situationen.

I næste artikel vil vi lege lidt med databasen, fylde nogen data i, crashe databasen, tilføje nye noder og gennemgå nogle af de forskellige teknikker du kan bruge for at holde øje med hvad din database laver.

2) Hvis du stadig siger "hvis det går galt", har du ikke arbejdet med drift længe nok endnu :-)

Om forfatteren:

Mikkel Mikjær Christensen arbejder til dagligt med opsætning, optimering og sikring af professionelle hostingløsninger baseret på Opensource igennem sit eget firma Mikjaer Aps (https://www.mikjaer.com), Mikkel blogger til dagligt på sin egen blog http://www.specialhosting.dk hvor han kommer med gode råd og guides relateret til sit daglige arbejde med Linux og andre Opensource systemer.

Pointere fra C til Java

1. Fra fordums storhed...

Der er næppe noget datalogisk begreb, der i den grad har givet anledning til svedige håndflader og nervøse trækninger omkring øjnene, end pointere.

Sproget C har fra tidligste tider haft (og har) en afgrænset sæt af datatyper til at udtrykke viden om data som programmet skal kunne operere med.

I de følgende sider går vi ud fra, at vi befinder os på en 32-bit maskine. I realiteten kunne det lige så godt være en 64-bit maskine, men dér spiller nu alligevel mange andre forhold ind, som gør det lidt mere kompliceret.

.....

En pointer er i sig selv blot en heltals-type (en long int for at være præcis). Udvikleren kan nu ændre pointerens gemte 32-bit heltal til at pege på en ny lager-celle.

```
int* ptr = 0L;
ptr = &minVar;
```

Skal der mere komplekse strukturer på banen, er vi typisk nødt til at udtrykke det ved hjælp af kald til kerne-bibliotekerne for at få allokeret memory på runtime:

```
int* ptr = 0L;
ptr = (int *) malloc(4 * sizeof(int));
```

Hvis memory er oprettet dynamisk, skal den afmeldes således at OS'et kan genbruge memoryblokken, eller risikerer vi memory leak.

Pointere er anvendelige, da de er yderst fleksible, og kan i princippet udpege en memory-celle overalt i ram'en. De har dog også en række ulemper...

1. Syntaktisk er de simple, men pointeren er uvidende om **hvad** de peger på. Sproget antager, at programmøren ved det, hvilket ikke nødvendigvis er tilfældet. Der er derfor brug for typecasts, der ikke nødvendigvis garanterer, at konverteringen er meningsfuld.
2. Da pointere er heltal, kan de opereres på som sådanne. Semantikken er imidlertid ikke altid klar addition og subtraktion er meningsfulde, men hvad menes med 2 * pointeren ? Hvis pointeres rammer det område af heap'en hvor kernen holder sine data, terminerer programmet.
3. Da arrays i C næsten altid implementeres som pointere, risikerer vi at komme til at skrive ud over slutningen af arrayen og derved (hvis vi er heldige) overskrive en funktions returadresse eller tilføje malware-kode placeret i protected memory (buffer overruns)
4. Hvis pointere anvendes til at udpege funktioner eller andre pointere, kræver det sort kaffe og et koldt hovede at visualisere datas repræsentation, endsige hvorledes pointeren skal tildeles en værdi.

2. ... til en ny plads i "Solen"

Nu hedengangne Sun Microsystems (R.I.P) puslede omkring 1990 med ideer om at kunne placere kode i alt fra store servere til små kaffemaskiner. Her ville C ikke være specielt anvendeligt, da pointere netop kan forårsage utilsigtede termineringer af kaffemaskinens styrekode, dvs. ingen kaffe (!). Da dette klart er uacceptabelt, vendte de bøtten i vejret med argumentet: hvad nu hvis programmet ikke kører ovenpå den fysiske CPU, men der imod på en simuleret CPU? Så kunne den simulerede CPU startes/stoppes uden at hardwaren blev påvirket. Den Virtuelle Maskine var født!

Javas virtuelle maskine (JVM) er en simulering af en simpel CPU med et par registre og et mindre kommandosæt. JVM udgør for java-programmets synsvinkel computeren. Java kan derfor ikke direkte tilgå hardwaren, men er nødt til at få JVM til at forhandle med OS'et på dets vegne. Hermed opnår vi en række fordele...

1. JVM kan antage at programmøren er et rodehovede, der ikke kan finde ud af at bruge memory korrekt, så den virtuelle maskine må nok hellere sørge for selv at holde styr på allocation/deallocation.
2. JVM bliver nu fodret med java bytecodes (assembler-koden til den virtuelle CPU) og vil derfor kunne afgøre, indenfor visse grænser, om der ville ske utilsigtede side-effekter inden koden bliver udført. Typisk er det stack-overflow eller out-of-memory JVM kan checke for ved hjælp af indbyggede heuristikker. Hvis JVM kan afgøre, at koden er usikker, nægter den at starte programmet.

Fra et Java-programs synsvinkel "stopper verden" i den virtuelle maskine. Derfor bliver variable også oprettet i hvad programmet tror er JVM'ens adresserum. JVM sørger dernæst for at der bliver oprettet den nødvendige plads i den fysiske RAM. Når variable ryger ud af scope, sørger JVM'en selv for garbage collection. Det er meget svært at lave memory leaks i java, men med en smule omtanke og indsats **kan** det faktisk godt lade sig gøre.

Sun annoncerer at pointere ikke findes i Java ...

Sun annoncerede fra starten med at java ikke havde pointere længere. Ved nærmere inspektion viste det sig at være en sandhed med visse modifikationer. Pointerne var der stadig, men de blev opfattet anderledes end i C. Man kaldte det nu en reference i stedet for.

En reference er en datastruktur, der typisk indeholder memory-references og en angivelse af typen på det udpegede element. Da sproget både er compileret og fortolket, betyder det at en variabels type-information er tilgængelig på runtime.

Det giver programmøren mulighed for at spørge en variabel om hvorvidt den peger på et heltal eller en bil (!). Java er jo dybt objekt-orienteret, så programmøren har indflydelse på pragmatikken om en bil giver mening, og hvad det lige er man forstår ved en sådan:

```
If (minObjekt instanceof Car) { .... }
```

Hvis klassedefinitionen er syntaktisk acceptabel for compileren, kan vi nu operere med objekt-variable på samme måde som vi opererer med primitive typer:

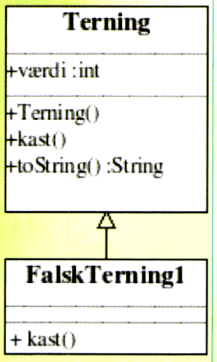
```
Car mingolf = new Car("Volkswagen", "golf");
// .....
mingolf = null;
```

Bemærk oprettelsen med new() operatoren. Instantieringen kræver klasse-navnet to gange, da type-erklæring og instantiering er to forskellige ting i java.

Objekt-nedlæggelsen er yderst simpel glem alt om objektet og antag at JVM får garbage-collected det. Det er altså ikke programmet selv, der slipper af med memory den opgave er overladt til en bagved kørende tråd, garbage collectoren, der traverserer JVM allokerede memory med en vis periodicitet.

Dette eksempel bør ikke overføres til C eller C++ da vi får øjeblikkelig memory-leak (årsagen hertil er simpel og overlades som en let øvelse til de få læsere, der har kunne holde sig vågne indtil videre!)

Referencer anvendes også i forbindelse med nedarvning. Da Java i sin natur er rent objekt-orienteret, vil programmøren have mulighed for at håndtere variationer over en klasse ved at bruge nedarvning (inheritance). En terning (som fx. er en del af et spil) ses her, findes i en variant som er falsk - den falder på en anden måde når den kastes:



(Fra Jacob Nordfalck: http://javabog.dk/OOP/kapitel5.jsp)

Det er imidlertid vigtigt at holde sig for øje, at instantiering af en ***FalskTerning*** indebærer oprettelse af referencer også til et Terninge-objekt, der igen vil referere til objekter af sin superklasse indtil vi når rod-klassen Object. Dette er en dyr proces udtrykt i plads- og tidsforbrug, men er en naturlig konsekvens af, at en ***FalskTerning*** i sin natur er en ***Terning***, omend en specialiseret udgave af samme.

Objekt-instantiering vil efterfølgende virke på normal vis:

```
FalskTerning tern = new FalskTerning();
tern.kast();
```

Det ville imidlertid være lige så korrekt at udtrykke det som

```
Terning tern = new FalskTerning();
tern.kast()
```

I sidste eksempel bestemmes funktionaliteten på runtime (sen binding). Begge terninge-typer kan svare på beskeden ***kast()***, men da croupieren jo helst ikke skulle opdage, at vi har brugt en falsk terning, vil den konkrete implementering automatisk vælges udfra den reference programmet har på runtime, ikke hvad koden syntaktisk har fået fået at vide på compiletime.

Alle steder hvor vi kunne tænkes at bruge en normal terning (superklasse-objektet) har vi lov til at erstatte objektet med et objekt af en subklasse, hvorefter funktionaliteten vælges på kørselstidspunktet.

Fordelene med denne mekanisme er måske ikke umiddelbar synlig, men den giver os en mulighed for at få referencer til at følge vilkårlige subklasser i de situationer, hvor vi ikke nødvendigvis kan bestemmer på compile-time hvilket subklasse-objekt, der er meningsfuld. Som oftest er det i de situationer, hvor der er en tidsdimension involveret.

Dette kaldes under eet for polymorfi og er en af grundpillerne i alle moderne objekt-orienterede sprog.

Pointen

Pointere er en nødvendighed i alle programmeringssprog - det er måden at hente værdien af de data, maskinen har i hukommelsen, som man siger med slet skjult antropomorfisme. Men objekt-orienterede sprog pakker denne funktionalitet ind på måder, så man ikke så let kommer til at misbruge den. Til forskel fra skydevåben er de eneste pointere, som er farlige, dem, som rammer ved siden af.

Om forfatteren Hjerre F. Hviid

Hjerre er bl.a. INTp, (hvilket bl.a. er en betegnelse for programmører der elsker at programmere og en forkortelse for introversion, intuition, tænkning, perception). Han er også en empatisk kyniker, som elsker ordspil og ordkløveri. Desuden er han henfalden til sort humor, absurditeter, klassisk musik, katte og Kierkegaard.

Hjerre er desuden specialiseret inden for UNIX, Linux, C/ C++, Java, Perl, X11, Motif, KDE, databaser og SQL. Han er ophavsmand til begreberne Jørgen Clevinsk overdrive og foretager gerne et 10.000 liniers eftersyn for C/C++-programmører.

Han er uddannet datalog (med psykologi) fra AUC. Endvidere smykker han sig med titlen Open Source programmør – med udtalt forkærlighed for hurtige pingviner, røde smådjævle og grønne æbler.

Du kan finde Hjerre på Facebook.

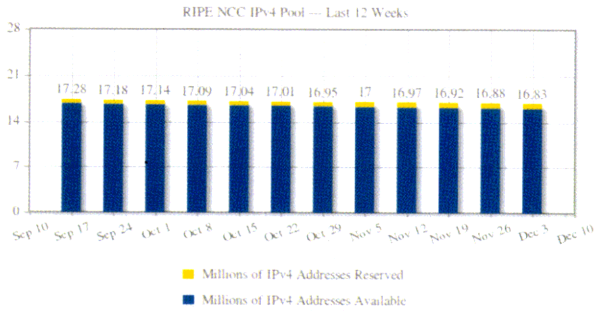
❍

IPv6 Status

Antallet af IPv4 adresser i Europa falder

RIPE (Reseaux IP Europeenne, Research IP Europe) begyndte at uddele adresser fra den sidste gruppe adresser, det sidste /8 blok. En /8 blok indeholder 256*65536 adresser og kan splittes op i mindre blokke, hvilket er indlysende nødvendigt på nuværende tidspunkt.

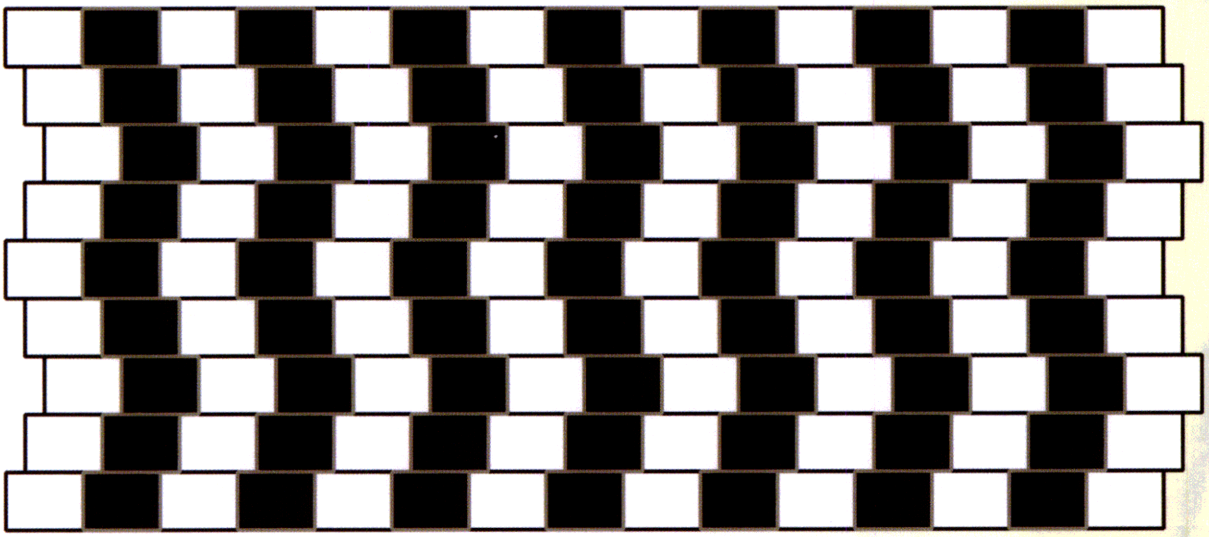
Der er ingen umiddelbar fare for adressemangel i Danmark, idet TDC anslår at man har nok til de næste 6-7 år, og TDC er begyndt at indkræve ubrugte adresser m.v.



På www.ripe.net kan man følge med i uddeling af IPv4

Ikke desto mindre vil det være en fordel for danske virksomheder at have IPv6, så kunder i fjernøsten i nær fremtid kan hente informationer om danske produkter.

❍



GNU parallel

GNU Parallel – parallel power til din kommandolinje

Ole Tange <ole@tange.dk>

Som kommandolinjebruger kommer man ofte ud fra at skulle udføre den samme opgave på en stribe filer/brugere/servere/tabeller. Ofte er man ligeglad med hvilken, der bliver behandlet først, og det vil være helt fint at udføre dem i parallel – men de kan ikke alle køre i parallel samtidigt, da det vil overbelaste din computer.

Det er den type problemer, som GNU Parallel kan hjælpe med at løse.

I denne artikel er der eksempler på brug af GNU Parallel. Eksemplerne er holdt simple for at gøre det nemt at forstå ideen, men der er intet i GNU Parallel som forhindrer dig i at bruge GNU Parallel til mere komplekse opgaver. Eksempler på mere komplekse opgaver kan findes i manual siden for parallel.

Dit første parallel job

Start med at installere GNU Parallel. GNU Parallel eksisterer som pakke for de fleste nyere UN*X distributioner. Hvis det ikke er åbenlyst, hvordan det installeres på dit system, så kik på www.gnu.org/s/parallel. Efter installation find en stak filer på din computer og gzip dem i parallel:

```
parallel gzip ::: *
```

Det, som sker her, er at din shell expanderer * til filerne i dit dir. ::: fortæller GNU Parallel, at den skal tage argumenterne fra kommandolinjen, og gzip er kommandoen, der skal køres. Jobene køres herefter i parallel. Efter du har gzippet filerne, så lad os genpakke dem med bzip2:

```
parallel "zcat {} | bzip2 >{.}.bz2" ::: *
```

Her bliver {} erstattet med filenavnet. Outputtet fra zcat pipes ind i bzip2 som pakker outputtet. {.} erstattes med filnavnet uden extension (e.g. foo.gz bliver foo), så output fra file.gz bliver gemt i file.bz2.

GNU Parallel tvinger dig ikke til at quote hele kommandoen, så du kunne lige så godt have skrevet:

```
parallel zcat {} "|" bzip2 ">"{.}.bz2 ::: *
```

Kun shells specialtegn skal quotes (da de ellers bliver fortolket af shellen).

Læsning af input

Vi har allerede set, at input kan gives på kommandolinen. Input kan også pipes ind i GNU Parallel:

```
find . -type f | parallel gzip
```

GNU Parallel bruger newline som record separator og har ingen problemer med filnavne, der indeholder mellemrum (space), ' (single quote) eller " (double quote). Hvis du har normale brugere på dit system, så har du sandsynligvis oplevet filenavne med de tegn. Hvis du har brugere, som er rigtigt onde og benytter newline i filnavne, så kan du bruge NULL som record separator:

```
find . -type f -print0 | parallel -0 gzip
```

GNU Parallel kan læse fra en fil med -a:

```
parallel -a filelist gzip
```

I stedet for -a kan du bruge :::: (fire koloner):

```
parallel gzip :::: filelist
```

Du kan bruge :::: og ::: flere gange. Så laves alle kombinationer, og du skal bruge {tal} som erstatningsstreng:

```
parallel cat {2} '|' gzip -{1} '>' \
{2.}.{1}.gz ::: 1 5 9 :::: filelist
```

Her laves 3 gzippede versioner af alle filer, en -1, en -3 og en -9.

Hvis dit input er i kollonner kan du splitte med --colsep:

```
cat filelist.tsv |
parallel --colsep '\t' diff {1} {2} ">" {3}
```

`--colsep` er et regexp, så du kan match mere avancerede kolonneseparatorer.

Byg kommandoen

Lige som xargs kan GNU Parallel tage flere input linjer og putte dem i den samme kommando. Sammenlign disse:

```
ls *.gz | parallel mv {} arkiv
ls *.gz | parallel -X mv {} arkiv
```

Den første vil køre mv for hver .gz-fil, mens den anden vil putte så mange filer som muligt ind i {} før kommandoen køres.

`{}` kan stå hvorsomhelst i kommandoen, men hvis den er del af et ord, vil det ord blive gentaget, hvis man bruger `-X`:

```
(echo 1; echo 2) | \
parallel -X echo foo bar{}baz quux
```

vil gentage bar-baz og skrive:

```
foo bar1baz bar2baz quux
```

Hvis du ikke giver GNU Parallel nogen kommando, vil GNU Parallel antage, at alle inputlinjer er kommandolinjer og køre dem i parallel:

```
(echo ls; echo grep root /etc/passwd) | parallel
```

Kontrol af output

En af udfordringerne ved at køre jobs i parallel er at sørge for, at output fra de forskellige jobs ikke bliver blandet sammen. traceroute er et godt eksempel på det, da traceroute langsomt skriver dele af linjer ud. Prøv:

```
traceroute foss.org.my & \
traceroute debian.org & \
traceroute freenetproject.org & wait
```

og sammenlign det med outputtet fra:

```
parallel traceroute ::: foss.org.my \
debian.org freenetproject.org
```

Som du kan se skriver GNU Parallel først outputtet ud, når et job er færdigt. Derved sikres at output fra forskellige jobs aldrig blandes. Hvis du insisterer, kan GNU Parallel ved brug af `-u` give dig outputtet med det samme, men det betyder at output fra forskellige jobs kan blandes sammen.

Nogle gange er det vigtigt at rækkefølgen af output er den samme som input. Det klarer `-k` for dig:

```
parallel -j4 -k echo {}';' sleep {} ::: 3 2 1 4
```

Dette kører alle 4 jobs i parallel, men output af 2 og 1 afventer, at det første jobs bliver færdigt.

Udførelse af jobs

GNU Parallel kører som udgangpunkt 1 job per cpu core. Du kan ændre dette med `-j`. Et heltal angiver antallet af jobs i parallel (fx `-j 4` for 4 jobs i parallel), mens en procentværdi ganges med antallet af cpu cores (e.g. `-j 100%` for at køre 1 job per CPU core):

```
parallel -j100% gzip ::: *
```

Hvis du giver -j et filnavn, bruges indholdet af filen som parameteren:

```
parallel -j /tmp/number_of_jobs_to_run gzip ::: *
```

Filen læses efter hvert job afslutter. Dermed kan du ændre antallet af jobs, mens GNU Parallel kører. Det er specielt brugbart hvis der er tale om mange jobs over lang tid, og det bliver nødvendigt at prioritere andre processer på computeren.

For at se de jobs, der kører lige nu, kan du sende GNU Parallel `SIGUSR1`:

```
killall -USR1 parallel
```

GNU Parallel udskriver da de kørende jobs på `STDERR`.

Hvis du fortryder at have startet en masse jobs kan du bare trykke CTRL-C. Men du vil være sikker på ikke at have halvfærdige jobs, bør du istedet sende `SIGTERM` til GNU Parallel:

```
killall -TERM parallel
```

Det fortæller GNU Parallel, at den ikke skal starte nye jobs, men i stedet vente på at de kørende jobs bliver færdige.

Kommandoen, der er kørt, kan udskrives før output med `-v`:

```
parallel -v gzip ::: *
```

Argumentet, der blev brugt, kan udskrives før hver linje med `--tag`:

```
parallel --tag ls -l ::: *
```

For at følge udførelsen kan du bruge --progress and --eta:

```
parallel --progress gzip ::: *
parallel --eta gzip ::: *
```

Det er specielt brugbart, når jobs kører på andre maskiner.

Andre maskiner

GNU Parallel kan køre jobs på andre maskiner. Dermed kan du bruge CPUerne på andre maskiner til at udføre behandlingen. Som eksempel vil vi bruge ompakning af .gz-filer til .bz2-filer, men du kan lige så nemt udføre andre beregningstunge jobs så som video-encoding eller billedbehandling.

Du skal kunne logge ind på de andre maskiner med `ssh` uden password. (her kan `ssh-agent` være praktisk). Hvis du har behov for at overføre filer, skal `rsync` være installeret, og hvis GNU Parallel selv skal finde ud af, hvormange CPU'er hver computer har, skal GNU Parallel også være installeret på de andre maskiner.

Prøv dette simple eksempel for at kontrollere, at dit setup virker:

```
parallel --sshlogin yourserver.example.dk\
 hostname';' echo {} ::: 1 2 3
```

Dette skulle gerne udskrive navnet på din server 3 gange fulgt af tallene 1, 2 og 3. `--sshlogin` kan forkortes til `-S`. For at køre på mere end een server kør:

```
parallel -S\
 server.example.dk,server2.example.net \
 hostname';' echo {} ::: 1 2 3
```

Hvis du har et andet login, kan du angive det som `login@` foran servernavnet – præcis som du ville gøre med `ssh`. Du kan angive flere `-S` istedet for at benytte `,` (komma):

```
parallel -S yourserver.example.com -S \
 mylogin@server2.example.net hostname';' \
 echo {}  ::: 1 2 3
```

Det specielle sshlogin `:` er din lokale maskine:

```
parallel -S yourserver.example.com -S\
 mylogin@server2.example.net -S : hostname';'\
 echo {} ::: 1 2 3
```

Her kan du komme ud for, at GNU Parallel kører alle 3 jobs på din lokale maskine, fordi jobbene er så hurtige at udføre.

Hvis du har en fil med en liste af sshlogins, kan du bede GNU Parallel bruge den fil:

```
parallel --sshloginfile mylistofsshlogins
hostname';' echo {} ::: 1 2 3
```

Filen skal dog ligge i ~/**.parallel**. Det specielle sshlogin `..` (to punktummer) læser sshloginfilen `~/.parallel/sshloginfile`:

```
parallel -S .. hostname';' echo {} ::: 1 2 3
```

Overførsel af filer

Hvis dine servere ikke deler filsystem (med NFS eller lignende) er der ofte behov for at overføre filerne, der skal behandles, til de andre maskiner, og efter behandlingen overføre resultatet tilbage til den lokale maskine.

Brug `--transfer` til at overføre til de andre maskiner:

```
parallel -S .. --transfer gzip '< {} |\
 wc -c' ::: *.txt
```

Her overfører vi hver .txt-fil til de andre maskiner, pakker dem og tæller antallet af bytes, som den pakkede udgave fylder.

Efter hver overførsel vil du normalt fjerne de overførte filer fra de andre computere igen. Det klarer **--cleanup** for dig:

```
parallel -S .. --transfer --cleanup \
 gzip '< {} | wc -c' ::: *.txt
```

Behandling af en fil giver often en resultatfil, som du gerne vil have kopieret tilbage, hvorefter både den overførte og resultatfilen skal slettes på de andre computere:

```
parallel -S .. --transfer --return {.}.bz2\
 --cleanup zcat '< {} | bzip2 >{.}.bz2' ::: *.gz
```

Her bliver .gz-filer overført. Derefter bliver de genpakket med **zcat** og **bzip2**. Efter det er gjort bliver den resulterende .bz2-fil kopieret tilbage, og .gz- og .bz2-filen slettes til slut på de andre computere. Kombinationen **--transfer --cleanup --return foo** bruges så ofte, at den har fået sin egen forkortelse: **--trc foo**

Du kan give flere **--trc** hvis din kommando giver flere resultatfiler.

GNU Parallel som del af et script

Jo mere du træner brug af GNU Parallel desto flere steder vil du se, at det kan bruges. Hver gang du skriver en for-løkke eller en while-read-løkke, så overvej om dette kunne gøres i parallel. Ofte kan for-løkken helt erstattes med en enkelt linje, der bruger GNU Parallel; hvis jobbet er meget komplekst, kan det være nemmere at skrive et script og kalde det script fra GNU Parallel. Men der vil stadig være tilfælde, hvor for-løkken er så kompleks, at ingen af disse er en mulighed.

Men måske kan du bruge **parallel --semaphore** i stedet. **sem** er et alias for **parallel --semaphore**.

```
for i in `ls *.log` ; do
        [... en masse komplekse linjer her ...]
        sem -j4 --id my_id do_work $i
done
sem --wait --id my_id
```

Dette vil starte **do_work** i baggrunden indtil 4 jobs kører. Næste gang **sem** bliver kaldt, vil den vente på at et af de 4 jobs bliver færdige, før endnu et job startes. Den sidste linje

```
sem --wait
```

venter indtil alle jobs startet af sem er færdige. **my_id** er en unik streng, der bruges af **sem** til at idenficere dette script – du kunne jo have andre sem's til at kører samtidigt. Hvis du kun kører sem fra eet script af gangen, behøver du ikke **--id my_id**.

sem bruger normalt **-j1** og virker derved som en mutex. Det er brugbart hvis du kun vil have een instans af programmet til at køre ad gangen.

GNU Parallel som en jobkø

Med få linjers kode kan GNU Parallel virke som en jobkø:

```
true >jobqueue; tail -f jobqueue | parallel
```

For putte et job i køen:

```
echo my_command my_arg >> jobqueue
```

Du kan naturligvis bruge **-S** til at distribuere jobs til andre maskiner:

```
echo >jobqueue; \
tail -f jobqueue | parallel -S ..
```

(***echo >jobqueue*** sletter hvad der måtte være af indhold i jobqueuefilen.)

Hvis du har et dir hvori brugere putter filer, som automatisk skal behandles, kan du gøre dette, hvis du kører GNU/Linux:inotifywait -q -m -r -e CLOSE_WRITE \

```
 --format %w%f my_dir | parallel -u echo
```

Kommandoen, der kører her, er **echo**. Det bliver udført på hver fil, der puttes i my_dir eller et underdir af **my_dir**. Også her kan du naturligvis bruge **-S** til at køre jobbene på andre computere:

```
inotifywait -q -m -r -e CLOSE_WRITE \
 --format %w%f my_dir | parallel -S ..  -u echo
```

Historien bag GNU Parallel

GNU Parallel startede ud som to separate programmer: **xxargs** og **parallel.** Formålet med **xxargs** var at virke lige som **xargs**, men uden de ubehagelige overraskelser ved space, ' og " (space, single-quote og double-quote). Formålet med **parallel** var simpelthen at køre jobs i parallel. Udviklingen af begge programmer startede omkring 2001. Jeg fandt ud af, at de to programmer ofte blev brugt sammen, så i 2005 bliv de samlet som parallel og navnet **xxargs** forsvandt.

Parallel havde derfor ikke eet men to mål: At overflødiggøre xargs og at køre kommandoer i parallel.

I 2010 blev Parallel et officielt GNU-værktøj og navnet blev ændret til GNU Parallel. En kort instruktionsvideo i brugen af GNU Parallel kan findes på: http://pi.dk/1.

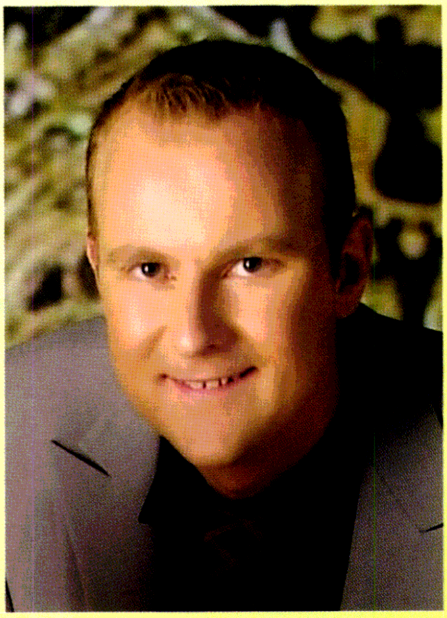
Få hjælp på mailinglisten

Jeg håber, du har tænkt på situationer, hvor GNU Parallel kan være til gavn for dig. Hvis du kan lide GNU Parallel, så fortæl andre om det gennem email lists, forums, blogs, og sociale netværk, eller køb GNU Parallels merchandise: https://www.gnu.org/software/parallel/merchandise.html

Hvis GNU Parallel sparer dig for penge, så donér til FSF https://my.fsf.org/donate. Hvis du har spørgsmål om GNU Parallel, så hop på mailinglisten på: http://lists.gnu.org/mailman/listinfo/parallel

Om forfatteren

Ole Tange arbejder inden for bioinformatik i København, men er åben for nye udfordringer. Han er aktiv i miljøet omkring fri software, og er bedst kendt for "den patenterede webshop" (http://ole.tange.dk/swpat), som illusterer problemerne ved software patenter. Han kan nås på: <ole@tange.dk>.



Slangetæmmer i det moderne Danmark

Dette er ikke en annonce

Af Jon Bendtsen og Donald Axel

Jon har afprøvet en netværks induktionsmåler i et rimeligt prisleje, nemlig FlukeNetworks' Intellitone Pro-toner og Probe.

Pro-toner, main-station, som ses på billedet nedenfor, forbindes til netstikket, typisk fra kontoret, hvorfra man ønsker at spore forbindelsens videreførelse.

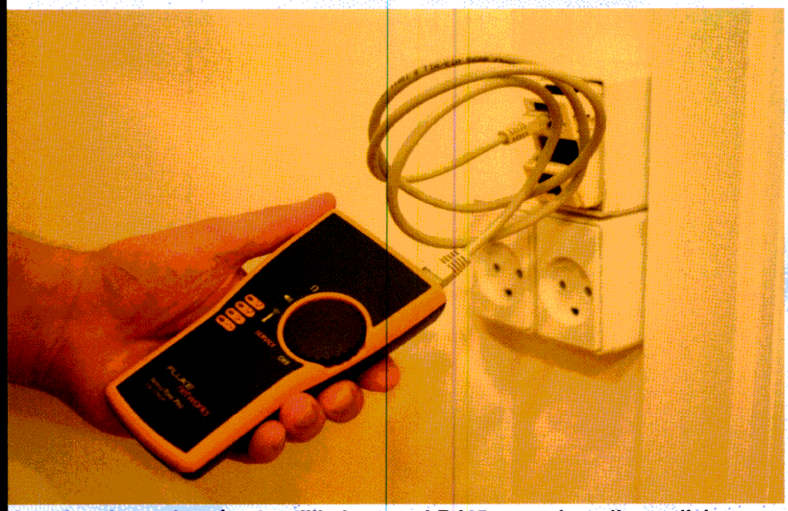
Når den er forbundet, lader man den udsende et prøvesignal, digitalt eller analogt og så er det muligt med den anden enhed, proben, som ligner en slags TV-fjernbetjening eller måske en elektrisk tandbørste, at afsøge ledningerne i krydsfeltet for at finde den eller de ledninger, som fører til main-stationen (Pro-toner'en).

Proben kan "se", hvilke ledninger der er aktive og i forbindelse med pro-toneren fordi den kan kende bitmønsteret.

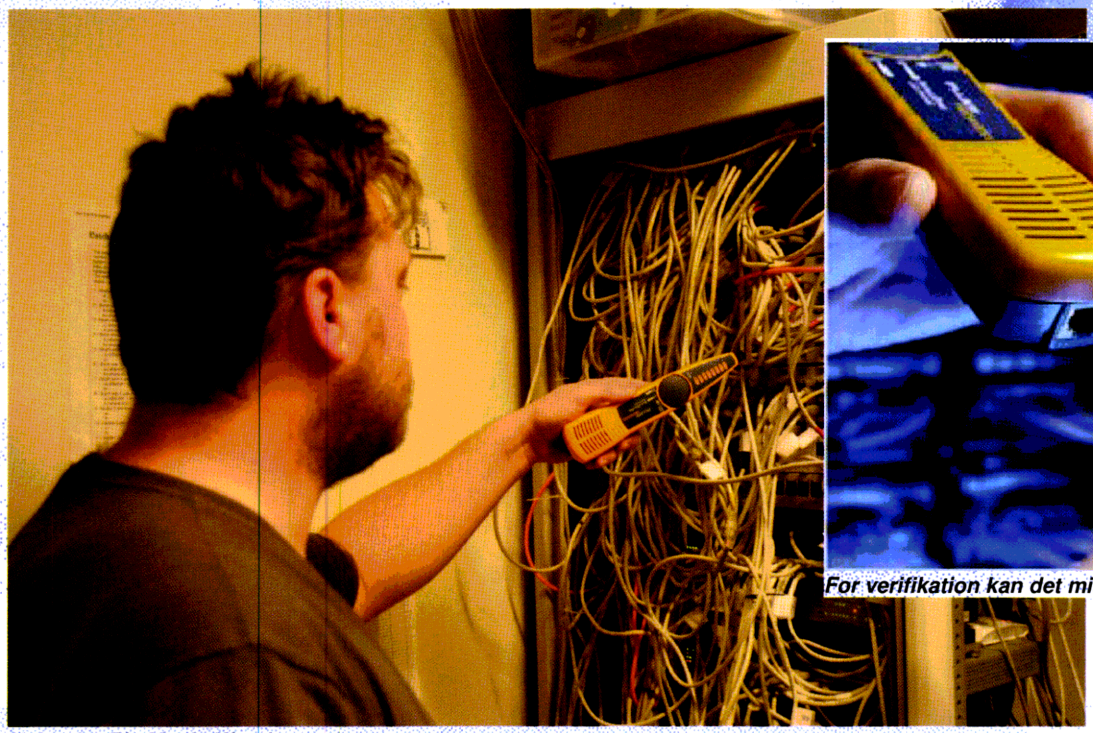
Den har to indstillinger: Med den ene kan man nærme sig et kabelbundt; når den bipper, er man i nærheden af den rigtige ledning. I den anden indstilling gennemgår man kablerne i bundtet én for én - med berøring af spidsen - og proben siger BEEP når man rører ved den rigtige. På billedet ses personen Hairy JonB med sin "Magic Wand" eftersøge ledningerne i et krydsfelt - desværre har han glemt at tænde for proben, men billedet er ellers selvdokumenterende.

Hovedstationen kan sende digitalt (10/100 MHz) og analogt - dermed har man et værktøj til flere situationer samlet på meget lille plads.

Glædelig jul til Netværksafdelingerne!



Hovedstationen består af en lille box med RJ45 og andre stik, og tilsluttes typisk på et kontor, hvorfra man vil spore forbindelsen ind til krydsfeltet



Hairy JonB's Magic Wand

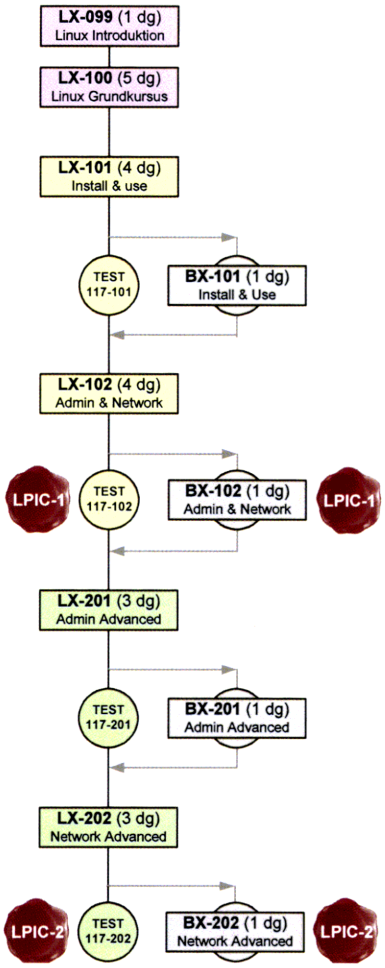
For verifikation kan det mistænkte kabel sættes i proben

SUPERUSERS

Linux

Kurser & Certificering

Linux Professional Institute



RØD PAKKE:
"I gang med Linux"

Kr. 18.900,- (listepris kr. 22.500,-)

Prisen dækker kurserne LX-099 og LX-100.

Indføring i filstrukturen, begreber, kommandoer, værktøjer, dokumentation, fil- og kataloghåndtering, processer, brugere, filrettigheder, shellen, wildcards, redirigering, pipes ... og meget andet.

GUL PAKKE:
LPIC-1:
"Junior Level Linux Certification"

Kr. 27.900,- (listepris kr. 33.200,-)

Prisen dækker kurserne LX-101 og LX-102 samt 2 test.

LPIC-1 er en grundlæggende certificering.
Fokus er på at arbejde fra kommandolinie, udføre simple administrative opgaver og kunne opsætte og forbinde en arbejdsstation.

GRØN PAKKE:
LPIC-2:
"Advanced Level Linux Certification"

Kr. 19.900,- (listepris kr. 25.800,-)

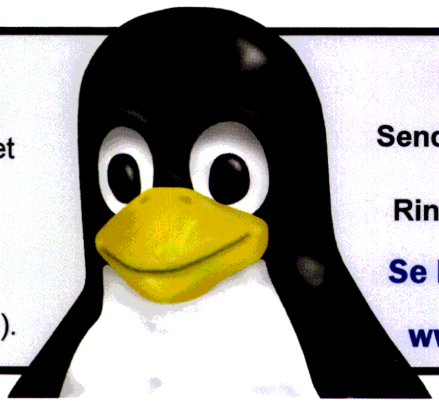
Prisen dækker kurserne LX-201 og LX-202 samt 2 test.

LPIC-2 er en overbygning på LPIC-1.
Omfatter administration af mellemstore installationer, med fokus på implementering, sikring, fejlsøgning og stabilisering af installationen.

Nr.	Kursus	Pris	Dg	Kort beskrivelse	Cert.
LX-099	Linux Introduktion	4.000,-	1	Hvis man ikke har arbejdet med andre operativsystemer, er dette kursus godt til at komme i gang med Linux og netværk.	Forudsætning for LX-100
LX-100	Linux Grundkursus	18.500,-	5	En grundlæggende og praktisk indføring i brug af Linux. Efter kurset kan du bruge Linux og det tilhørende netværk.	Forudsætning for LX-101
LX-101	Linux Install and Use	14.800,-	4	Kurset gennemgår grundlæggende installation, konfiguration og administration af en standalone Linux-client.	LPIC-1 117-101
LX-102	Linux Administration and Networking	14.800,-	4	Om installation og konfiguration af en Linux netværks-client og simple netværks-services på en Linux netværks-server.	LPIC-1 117-102
LX-201	Linux Administration Advanced	11.100,-	3	Konfig. af kerne, filsystemer og hardware, fildeling med hhv. NFS og SAMBA, logning samt automatisering af systemopgaver.	LPIC-2 117-201
LX-202	Linux Networking Advanced	11.100,-	3	Konfig. af netværk, email og news, DNS, Web Services, DHCP, NIS, LDAP, PAM, Linux System Security, routerkonfiguration, ...	LPIC-2 117-202

BX-kurser (Boot camps)
omfatter 1 dag hos SuperUsers med effektiv gennemgang af emnerne i det tilhørende LX- kursus.

Dagen afsluttes med den til kurset tilknyttede test (undervisning og test er inkluderet i prisen for BX-kurserne).



Sådan bestiller du:

Send en mail til: super@superusers.dk
eller
Ring til SuperUsers på: 48 28 07 06

Se komplette kursusbeskrivelser og datoer via www.superusers.dk/linux.htm

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**