Szerverkonszolidáció UML-el

Tomka Gergely

2004. április 22.

Egyetemünkön örökké jelen lévő, és egyre erősödő igény, hogy minden karnak, tanszéknek, intézetnek, helyi adminisztrátornak, egyébnek legyen Saját Szervere. Ez önmagában nem baj, sőt, pozitívum, hiszen örülünk annak, hogy az informatika betör az ilyen helyekre is, de számtalan problémát vet föl.

1. Megoldandó nehézségek

A felsorolt jelenségek minden egyetemi rendszergazda számára ismerősek. Vagy azért, mert tevőleges részese az ilyen helyzetek kialakulásának, és nem érti mi a baja ezzel a központi informatikának, vagy mert a központi informatikán dolgozik, és naponta többször is a haját tépi a helyzet alakulása folytán. A felsorolás távolról sem teljes, igyekeztem csak azokat említeni, amelyeket az UML bevezetése megold.

- Egy ilyen szórványszerver működhet olcsó, gagyi, megbízhatatlan hardveren. Ilyenkor lefagy, meg kell nézni mi a baja, kell keríteni bele gagyi hardverelemet, papírfecnivel kiékelni a cpu ventilátort, poroltót tartani a közelében, és a többi. Nem szerencsés megoldás.
- A szórványszerver lehet „state of the art” is, ami csak első pillantásra előny. Egy komolyabb szerver igen sok áramot fogyaszt, és igen sok meleget termel, ezért úgy a huszadik után már komoly költség a légkondicionálás és a szünetmentesek sem olcsók. Az erkölcsi hatásról nem is beszélve - én fizikailag rosszul vagyok egy telivér ibm szervertől, ha nem csinál semmit.
- A szórványszerver szórványrendszergazdával jár. Ritka a hozzáértő, akire tetszőleges erőforrást rá lehet bízni, és elvből benne sem bízunk igazán. A jó rendszergazda paranoiáját semmi sem korlátozza, a központi rendszergazdáét is csak a lustaság.
- Ellenőrzés és kontroll. Minél kényelmesebben és hatékonyabban lehessen információt gyűjteni, elosztani az erőforrásokat, ellenőrizni a felhasználókat, baj esetén megakadályozni az elharapózást. Ez sok kicsi, ismeretlen rootjelszavú rendszernél körülményes. Sőt.

E gondok megoldására, illetve enyhítésére számtalan megoldás létezik, jelen sorok írója ebből csak párat ismer, röviden jellemezzük őket, a teljesség kedvéért:

- „Apacs” jellegű virtual hostok: egyfelől igazán erőforrás takarékos, viszont nem minden szolgáltatás képes ilyenre, és ha a felhasználónak beleszólási jogot akarunk adni a konfigurációba, nincs könnyű dolgunk.

- Chroot jellegű környezetek: erőforrás takarékosak, de megvalósításuk körülményes, és az erőforrások (egyenlőtlen) elosztása körülményes, ha lehetséges egyáltalán.
- User Mode Linux: nem tökéletesen erőforrás takarékos megoldás, de teljes szabadságot ad a szórványrendszergazdának, az ellenőrzést és erőforrás-gazdálkodást gyerekjátékká teszi, és nem utolsó sorban, nagyon aranyos.

Vegyünk hát egy nagy levegőt, és fussuk át röviden, hogyan kell megvalósítani egy ilyen rendszert.

2. UML megvalósítás

Két élesen elkülönülő fázisra osztható. A host, az a gép, ahol az umlek futnak, és maguk az umlek két külön történet, és szerencsére elég kevés közük van egymáshoz. A továbbiakban simán umlnek nevezem ezt az entitást, sőt, magyarul fogom ragozni is, mert így kényelmes. Valamint, az eddigi hűvös, józan filozófiai megfontolások helyébe a tőlem megszokott "érdekes" megoldások lépnek, előre is elnézést kérek azért, hogy nem tudok értelmesen programozni, és hogy nem használok netről készen letölthető dolgokat. Én már csak ilyen vagyok...

Az alább részletezett megoldás pár alapelvre épül:

- Az umltől nem várunk nagy teljesítményt.
- A szórványrendszergazda teljes szabadságot élvez az umlen belül.
- A host szerver gazdája lusta.

2.1. A host szerver

Bármilyen, nekünk szimpatikus Linux rendszer lehet. Pár követelménynek meg kell felelnie, de egyik sem nehezen megvalósítható.

- TUN/TAP legyen a kernelben.
- 2.6.x kernel esetén kell egysoros patch, hogy működjön.
- Sok diszk.
- Sok RAM.
- Még diszk. Kell legyen jó nagy tmp könyvtár is.
- Uml-utilities csomag, Debian rendszeren. Máshol más lehet neve.

Jelen történetben ez egy IBM eServer xseries 335, két processzorral, 2 GB rammal, nem túl sok diszkkel. Tárterületet egy storage rendszer biztosít majd, mikor ez aktuális lesz. Az operációs rendszer Debian GNU/Linux, 2.6.x kernellel. Teljesen és tökéletesen elégedettek vagyunk vele. Mondjuk a kereskedőtől működésképtelenül érkezett, de azért vagyunk, hogy az ilyen problémákat megoldjuk.

2.2. Az uml

2.4.x sorozatú kernelből készül, az uml honlapról letöltött patchel, egyszeri kernelfordítással.

```
# make menuconfig ARCH=um
# make Linux ARCH=um
```

Ezután keletkezik egy `linux` bináris, ezt igény szerint strip-el kicsinyítsük le, és másoljuk oda, ahová jól esik. Ez egy futtatható bináris, ami elindít egy Linux kernelt. Haszna természetesen akkor van, ha megadunk neki egy fájlrendszert, amit ő root partícióként kezelhet. Lássunk pár parancssori opciót:

```
#/usr/local/bin/linux ubd0=rock.bin eth0=tuntap,tap6 \
umid=rock uml_dir=/var/lib/uml mem=128M con=pts
```

Az `ubd0` a root partíciót tartalmazó diszk-image. Elkészítése egyszerű, dd-vel egy megfelelő méretű image, losetup, mkfs, mount, debootstrap, chroot, utókonfig. Akinek ez a felsorolás nem mond semmit,forduljon hozzám bizalommal, nekem van egy nem tökéletes, de bőven elegendő 500 Mbyteos woody imagém, nagy örömmel közzéteszem. Bárhogy is csináljuk, az inittabban két dolgot jó átállítani. Az alt-ctrl-del hatása ne reboot legyen, hanem halt, ugyanis ez az egyetlen jel amit kényelmesen tudunk közölni az umllel kívülről, és irtsuk ki a tty-ket, úgysincs rá szükségünk.

Az `eth0` adja meg az umlből eth0 interfésznek látszó entitás külső megjelenését. Esetünkben ez egy ún. TUN/TAP eszköz, ami megfelelően beállítva olyan, mintha saját hálókártyája lenne a gépnek. Van még pár megoldás, és ez sem olyan egyszerű, később részletesebben is foglalkozom vele.

Az `umid` egy egyedi azonosítót ad az umlnek, az eredetileg járó krixkrax helyett. Ezzel lehet a ps kimenetében megkülönböztetni az umleket, és ez lesz a vezérlőfájlokat tartalmazó könyvtárnak is a neve. Az uml_dir opcióval megadott könyvtárban hozattatik létre a fönt említett vezérlőkönyvtár.

A `mem` opciót mindenki kitalálja, szerintem. Ezt amúgy nem lefoglalja fixen, hanem a tmp könyvtárban hoz létre egy fájlt, és abba írogat. Így ha éppen van ram szabadon, akkor ott csücsül az uml ramja a host gép cache-ben, és ekképpen gyors, ha nem, akkor meg nem. Szakértők szerint, ha nem akarunk túl lassú ramot, akkor ún. tmpfssel generált tmp könyvtárba helyeztessük ezeket a fájlokat. Ezt elhiszem, hogy igaz, de még nem értem, hogy miért.

A `con` opció adja meg, hogy hol keresse a konzolt. Eredetileg ez egy xterminál, de ez nem mutat jól egy szerveren, ezért ezt adjuk meg. Boot közben a stdoutra is ír, azt irányítsuk át valahová, a dev/nullban mindig van hely. Aki szereti látni a konzolt, annak sok lehetősége van, élvezze mértékkel.

A leállítás sem bonyolult. Kell hozzá az uml_mconsole nevű kis program, és tudnunk kell, hogy hol van a kontroll fájl. Ez a fenti példában így néz ki:

```
#uml_mconsole /var/lib/uml/rock/mconsole
(/var/lib/um) cad
```

Itt rögtön szemléltetem azt is, hogyan kell leállítani a megfelelően kiképzett umlt. Az mconsole cad parancsától az uml azt hiszi, alt-ctrl-delt nyomtak neki, és elmegy aludni. Tucatnyi más parancs is van, lehet kihúzni/bedugni eszközöket, megállítani az umlt (például mentés készítéséhez), le lehet lőni gondolkodás nélkül (konnektorkitépés szimulálásához).

2.3. Hálózat

Az umlek széles körű felhasználási lehetőségeihez mérten többféle hálózati interfész is lehet. Én kizárólag egyet használok itt és most, a TUN/TAP becenevűt. Ez némi scriptirogatás árán közel tökéletesen szabad felhasználást tesz lehetővé az umlen belülről. Ehhez természetesen jó sok mindent el kell hitetnünk a külvilággal. Például azt, hogy a host gépnek nem csak egy címee van, hogy a host gép egy hálókártyája több hálókártya, több MAC címmel, és természetesen a host gépen belül is el kell jutnia a megfelelő helyre a biteknek.

Be kell állítani egy tun/tap eszközt, és ezt a `tapN`-et kell megadnunk az umlnek, mint a hálózat felé eső végét. Ezen felül kell még sok minden, routolni kell az uml nyilvános ip címére igyekvő csomagokat a host gép `tapN` interfésze felé, statikus arptábla-bejegyzést is kell csinálni, stb. Erre nekem van egy scriptem, a függelékek között szerepel majd.

Az umlek tudnak csak egymással beszélgetni. Ennek legtöbb lehetőséggel kecsegtető módja az uml_switch nevű kis program. Megfelelő parancssori opcióval ebbe "beledughatjuk" az umleket, akik így látják egymást. Az uml_switch viselkedhet hubként és switchként is. Nekem erre most nincs szükségem, de nagyon kellemes kis "rongálható" kabinetet föl lehet építeni egy hostgépen futó sok umlből, melyek egy kijelölt speciális tűzfal-umlen és egy tun/tap eszközön át kapcsolódnak a külvilághoz. Egy fájlt egyszerűbb visszamásolni, mint egy tönkrement valós gépet újrainstallálni.

3. Karbantartás és ellenőrzés

3.1. Fájlrendszer, durva hibák

A rendszerünk egy önálló fájlban húzódik meg, amit blokkeszközként kezel. Ezért ha sérül, akkor ugyanúgy kell ellenőrizni is. Losetuppal blokkeszközzé varázsoljuk, fsckval ellenőrizzük. Ha például rootjelszóelfelejtés esete forog fenn, akkor föl is mountoljuk, és kedvünkre hegeszthetjük. Természetesen ilyen műveletek előtt jó leállítani az umlt. Esetleges betörés esetén hasznos, hogy az uml „standby” állapotba hozható, kimerevíthető. Eképpen tetten érhetjük a behatolót, átvizsgálhatjuk a memóriát, a fájlrendszert, és a behatolónak esélye sincs adatokat törölni, hiszen két órajelciklus között megtorpantunk.

Nagyon szeretem, hogy password recoveryhez nem kell odaszaladni a géphez, álldogálni a hideg és zajos szerverszobában, ne adj isten vészhelyzetben betaxizni éjszaka, hanem bárhonnan el tudom intézni.

3.2. Alkalmazások, használat módjának ellenőrzése

Az uml minden futó processzhez új threadot indít a hostgépen, és korrektül ki is tölti a megfelelő táblázatokat, ezért gond nélkül tudjuk követni, hogy a szórványrendszergazda éppen mit futtat.

```
# ps ax
31490 /usr/local/bin/linux (rock) [(tracing thread)]
...
31949 /usr/local/bin/linux (rock) [/sbin/syslogd]
...
6682 /usr/local/bin/linux (rock) [/usr/sbin/portsentry]
```

Látható, hogy ez az user nem érzi magát biztonságban. A lista nem teljes, ezért az nem látható, hogy az user még nem tökéletesen ura a technikának, például eszébe sem jutott az apache futó processzeinek számát csökkenteni. Sebaj, erre jó az uml. Játszótér.

Természetesen az az egyszerű tény, hogy az uml minden hálózati forgalma átmegy a hostgépen, ráadásul szépen szétosztva umlenként, roppant kényelmessé teszi a forgalom figyelését is. Sőt, az umlnek van olyan opciója, hogy logolja a szerveren belül történő dolgokat is, hogy az ssh feltörésével se kelljen bajlódnunk. Ezek a megfigyelési módok jogi kérdéseket is fölvetnek - ügyeljünk arra, hogy ártatlan játszadozásunk ne vonjon maga után börtönbüntetést. Ezt legegyszerűbben egy jól megfogalmazott házirenddel lehet elérni. Nekünk ilyen még nincs.

3.3. Tömeges használat

Kedvenc parancssoros környezetünkről van szó, tehát minden automatizálható, nincsen ez másképp az uml kérdéskörrel sem. Az már rám jellemző betegség, hogy ezek a scriptek nem oldanak meg minden megoldható problémát, de talán hasznos kiindulópont lehet a scriptírásilag kihívásokkal küszködőnek, és jó mulatság a témát ismerőknek.

A megoldás lelke egy felsorolás, minden uml információival:

```
agymosoda /var/lib/uml 192.168.0.3 192.188.242.118 128M
trinity /var/lib/uml 192.168.0.2 192.188.242.123 128M
able /var/lib/uml 192.168.0.4 192.188.242.119 128M
hok /var/lib/uml  192.168.0.5 192.188.242.116 128M
grable /var/lib/uml 192.168.0.6 192.188.242.115 128M
rock /var/lib/uml  192.168.0.7 192.188.242.114 128M
```

A 192.168.0.x cím az a tun/tap interfész belső címe, routolás végett. Ehhez tartozik indítóscript:

```
#!/bin/bash
grep $1 uml.list | (
# filenev, konyvtar, belso ip, kulso ip, memoria
read FILE DIR IIP OIP MEM
echo $IIP $OIP $DIR $FILE $MEM
echo "uml" $FILE "halozat cfg..."
TAP=‘tunctl -b‘
echo "ifcfg..."
ifconfig $TAP $IIP up
echo 1 > /proc/sys/net/ipv4/ip\_forward
echo "route..."
route add -host $OIP dev $TAP
echo 1 > /proc/sys/net/ipv4/conf/$TAP/proxy\_arp
echo "arp..."
arp -Ds $OIP eth0 pub
echo "ok."
echo $TAP > $DIR/$FILE.tap
echo "uml" $FILE "inditas..."
cd $DIR
export TMPDIR=/var/tmp
```

```
/home/tomka/linux ubd0=$FILE.bin eth0=tuntap,$TAP \
umid=$FILE uml_dir=$DIR mem=$MEM con=pts \
>/dev/null </dev/null &
        echo "ok" )
```

Hát mit is mondjak? Kicsit küzdünk a bash hiányosságaival, de egyszerű mint a faék. Eltesszük a tun/tap eszköz nevét, hogy leállításkor törölhessük. Leállítás:

```
#!/bin/bash
uml\_mconsole /var/lib/uml/$1/mconsole cad
echo "türelem"
sleep 10
tunctl -d ‘cat $1.tap‘
rm $1.tap
echo $1,", rest in peace "
```

Aludni kell 10 másodpercet, hogy az összes processz/thread leálljon. Türelem kérdése. Emiatt nem egyszerű leállítani a hostgépet. Itt két ponton is tettenérhető lustaságom - egyfelől nem keresek megoldást arra, hogy a shutdown hosszabb ideig várjon leállásnál az elhalásra. Másrészt meg mert nem ellenőrzöm, hogy tényleg leálltak-e az umlek, csak várok kicsit. Hát, sosem terveztem tökéletesnek lenni. Majd mikor először pórul járok a hostgép halomra gyilkolása miatt, majd megcsinálom rendesen, ígérem.

3.4. Sokszorosítás

Egy mindig készenlétben tartott "szűz" rendszerrel ez nem bonyolult feladat:

- `cp default.bin rock.bin`
- Megfelelő méretre hozzuk, a példa kedvéért 1 Gbyte-vel növeljük:
  - `dd if=/dev/zero bs=1024 count=1000000 >> rock.bin`
  - `losetup /dev/loop0 rock.bin`
  - `ext2resize /dev/loop0`
- `mount /dev/loop0/ /mnt`
- Szerkesztendő fájlok:
  - `/etc/hosts`, a hostnév kedvéért
  - `/etc/hostname`, dettó
  - `/etc/network/interfaces`, ipcím és társai
- `chroot .`, root és user jelszó változtatás
- `rm /etc/ssh/*key*`
- `dpkg-reconfigure ssh`, hogy egyedi ssh kulcsok legyenek
- `exit`
- `umount /mnt`
- `losetup -d /dev/loop0`
- `uml.list` szerkesztése, indítás

Ez nagyjából 4-5 perc, és főnökünk büszke lesz ránk.

4. Tények és tapasztalatok

4.1. Teljesítmény

Egyik szemem sír, a másik meg üveg. Vannak jó hírek, és rossz hírek. Kezdjük a rosszal.

Az uml a diszk-io teljesítményt nagyjából harmadolja. Ha a hostgépen 100 mbyte/sec sebességgel tudunk írni/olvasni, akkor az umlben 33 mbyte/sec lesz a maximális sebesség. Ez bizonyos szempontból kellemetlen, de így is gyorsabb, mint egy átlag hálózati kapcsolat. Mindenesetre a tanulság fontos - sok, nagy fájlt kiszolgáló szerverre keressünk más megoldást, esetleg nézünk körül a SKA patchek környékén.

A hálózattal már jobb a helyzet, erős gépen a 100 mbps-t kitöltötte, különösebb cpu terhelés nélkül. A CPU felhasználás esetén pedig egyenesen csodálatos a helyzet, a veszteség az UML miatt kevesebb mint 1%.

4.2. Megbízhatóság

Kiváló.

5. Zárszó

A rendszer nekem is, és a potenciális felhasználóknak elnyerte tetszését. Jelenleg ugyan csak hat darab fut, ezek közül az egyik napi több gigabyteos forgalmat bonyolít le (bulik képeivel), egy másikon közepesen komoly php/psql alkalmazás fut, hiba, zökkenő és fönnakadás nélkül. Tapasztalataim alapján jelen IBM 335 több tucat umlt is elbír a hátán, ami hatalmas előny most, mikor szerverkonszolidációt végzünk. A különböző bladeserverek világában elegáns megoldás rámutatni egy 1 unit magas szerkezetre, hogy "ott a szerverfarmunk".

Egy új uml életbe léptetése villámgyors folyamat, és ezzel hihetetlen mennyiségű piros pontot lehet szerezni minden értelemben. Minőségileg új kapcsolatot jelent a felhasználókkal, ha 10 perccel az igény bejelentése után már egy tökéletes kis szerver zakatol, ami csak az övék. Arról nem is beszélve, hogy a nagyarcú számolgatásoknál az uml kétszer is számít - a hostgép gazdája mondhatja lezserül, hogy "én karbantartok egy nagyvasat, persze azon fut vagy 30 szerver", de a szórványszerver felhasználója is mondhatja, hogy ő bizony karbantart egy szervert, ami kint van az Interneten, bizony. Mindenki jól jár.

Nekem külön öröm, hogy szabad szoftverekkel tudok olyan megoldást, és olyan új lehetőségeket nyújtani, melyeket más eszközzel csak összehasonlíthatatlanul drágábban és sokkal nagyobb energiaráfordítással lehet.

Általában, tapasztalataim szerint az User Mode Linux használata itt és most telt kecskéket, és érett, kártevőmentes káposztákat eredményezett. Csak javasolni tudom használatát.