Első lépések a Condor rendszerrel

Hogyan kezdjünk barátkozni ezzel a több platformon elérhető, elosztott számításokat lehetővé tevő rendszerrel...

A klaszterek használatának gondolata valamikor az 1990-es években született meg, amikor a hardverárak hirtelen esni kezdtek, a *PC*-k pedig egyre nagyobb teljesítménnyel bírtak. A cégek a valójában nagy méretű, ám mini-nek nevezett számítógépek használatáról kezdtek áttérni a már valóban kicsi és ezért „mikro" előtaggal illetett eszközök használatára. Ezzel párhuzamosan többen felismerték azt az érdekes ellentmondást, hogy az új, nagyobb teljesítményű gépek számítási kapacitása az idő nagy részében kihasználatlan marad, hiszen míg régen a teljesítmény egyetlen nagy dobozban volt jelen a cégnél, addig az új korszakban számos doboz között „fragmentálódott". Ma egy komolyabb vállalat asztali gépek százaival, vagy akár ezreivel rendelkezik, amelyek az idő túlnyomó részében kihasználatlanul állnak. Nem ritka ugyanakkor, hogy ugyanezek a vállalatok bizonyos feladatok elvégzéséhez, vagy egyszerűen csak versenyképességük megőrzéséhez komoly számítási teljesítményt igényelnek, vagyis egyszerre van jelen a rendszerben a pazarlás és az „éhezés". Éppen ez az az effektus, amely folyamatosan fenntartja manapság az érdeklődést a szuperszámítógépes megoldások és a klaszterek iránt.

Számos gyártó kulcsrakész klasztermegoldásokat kínál, persze nem föltétlen olcsón. Ugyanakkor szabadon használható, nyílt forrású eszközök segítségével az sem lehetetlen, hogy kihasználjuk az eleve meglevő kapacitásokat gyakorlatilag anélkül, hogy egyetlen fillért kellene költeni beruházásra. A megfelelő szoftverek segítségével megépíthetjük saját klaszterünket a saját gépeinkből. Ebben a cikkben egy a ***University of Winsconsin*** által kidolgozott ilyen megoldásról, nevezetesen a ***Condorról*** lesz szó.

A *Condor* alapötlete rendkívül egyszerű: telepítenünk kell valamennyi gépre, amit a klaszter részeként szeretnénk használni, a többi meg már megy magától. A *Condor* terminológiájában a klasztert *pool*-nak (kb. közös készlet) szokás nevezni, így ebben a cikkben a két kifejezés egyenértékűnek számít. A telepítés után a *pool* bármelyik gépéről indíthatunk számítási feladatokat (*job*). A *Condor* megvizsgálja a program igényeit, összeveti az aktuálisan rendelkezésre álló szabad erőforrásokkal, és oda irányítja a végrehajtást, ahol azt a leghatékonyabban oldható meg. Ha megtalálta a megfelelő gépet, a *jobot* áthelyezi arra, megvárja, amíg lefut, majd begyűjti az eredményeket. A *Condor* egyik legnagyobb előnye tehát az, hogy a klaszter használatához egyáltalán nem kell átírni a meglevő alkalmazásokat.

A gyakorlatban persze a dolog azért valamivel bonyolultabb. Először is a *Condort* a különböző gépekre másként kell telepíteni. Minden *Condor poolnak* van egy központi kezelője (***central manager***), amely – nevének megfelelően – a klaszter belső adminisztrációját végzi. Ez a gép figyeli, hogy a pool melyik tagján vannak szabad óraciklusok, és ez végzi a *jobok* igényeinek illetve az erőforrásoknak az összehangolását is. Egy *Condor pool* ezen kívül tartalmazhat úgynevezett ***„Submit"*** és ***„Full install"*** típusú gépeket. Az előbbi csoportba tartozók beküldhetnek futtatandó feladatokat, de maguk nem futtathatnak egyetlen ilyen programot sem. A másik csapatot értelemszerűen azok a gépek alkotják, amelyek mindkét művelettípusra fel vannak jogosítva.

Igények és telepítés

Ami a hálózatot illeti, a *Condor* használatához nincs szükség semmilyen új hálózati elemre, a már meglevő tökéletesen elegendő. A *Condor* számos különböző operációs rendszerrel képes együttműködni. Lehetőségeink a következők: *Linux*, *Solaris*, *Digital Unix*, *AIX*, *HP-UX*, *Mac OS X*, *MS Windows 2000* és *XP*. Ami a különböző hardver-architektúrákat illeti, ezen a téren is van néhány választási lehetőségünk: *Intel x86*, *PowerPC*, *SPARC* stb. Persze a rendszer működési logikájából az is következik, hogy az egy adott architektúrára fejlesztett *jobok* csak ugyanilyen gépeken lesznek képesek futni, vagyis egy *Intel x86*-ra fordított programtól ne várjuk el, hogy bármi máson fusson. Ebből pedig nem túl nehéz levonni azt a következtetést, hogy a *Condor* klasztereket érdemes egy adott architektúrából összeválogatni. Azért egy kivétel itt is akad: a Java alkalmazások történetesen a különböző architektúrák között is átvihetők, ezekre a fentiek tehát nem vonatkoznak.
Ebben a cikkben kizárólag a *Linuxra* való telepítésről lesz sz, mégpedig a úgy, hogy a platformfüggetlen tarlabdából indulunk el. Természetesen léteznek az egyes operációs rendszerekre illetve architektúrákra specifikus csomagok is, amelyekről bővebb információt a ➲ www.cs.wisc.edu/condor/downloads címen találhatunk.
Töltsük tehát le a platformfüggetlen csomagot erről a helyről, majd tömörítsük ki a következő paranccsal:

```
tar -zvf condor.tar.gz
```

A telepítéshez egyetlen dolgot kell tennünk: lefuttatni az ***sbin*** könyvtárban található `condor_install` nevű szkriptet. Mielőtt azonban így tennénk, hozzunk létre egy `condor` nevű felhasználót is a rendszeren. Biztonsági megfontolások miatt a *Condor* nem engedi, hogy bárhol a *root* nevében futtassunk programokat, így szükségünk lesz egy olyan közönséges felhasználói fiókra, amelynek nevében a dolgok bonyolódnak.
Az első kérdés, amit a szkript nekünk fog szegezni úgy hangzik, hogy hány gépből szeretnénk klasztert létrehozni. Ennek a kérdésnek igazán akkor van jelentősége, ha megosztott fájlrendszert használunk, ilyenkor ugyanis a szkript bekéri az összes gép nevét, majd a *Condor* telepítését azokon is automatikusan elvégzi. Ebben az esetben tehát ezekkel nekünk már nem kell foglalkoznunk. Ha nem használunk elosztott fájlrendszert, akkor a telepítést manuálisan kell mindenütt elvégezni. Ha Java alkalmazásokat is szeretnénk futtatni, akkor szintén telepítenünk kell mindenhova a Sun Java virtuális gépét is.
Az esetlegesen felmerülő problémákkal kapcsolatban maga a telepítőszkript is rengeteg segítséget nyújt, hiszen van súgója, illetve minden feltett kérdéshez tartozik magyarázó szöveg is. Ha pedig ezek nem segítenek, bármikor rendelkezésünkre áll a részletes felhasználói kézikönyv, illetve az alkalmazással foglalkozó levelezési lista.
A továbbiakban feltételezzük, hogy a `$CONDOR` változó tartalmazza annak a könyvtárnak az elérési útvonalát, ahova a *Condort* kibontottuk. Telepítés után a rendszert a következő paranccsal indíthatjuk el:

```
$CONDOR/bin/condor_master
```

Ez a parancs az összes olyan folyamatot elindítja, amelyekre a *Condor* működéséhez szükség van. Ez ezt jelenti, hogy a következő parancsot kiadva a központi kezelőn (*central manager*) összesen öt olyan folyamatot kell látnunk, amelynek neve a `condor_` előtaggal kezdődik:

```
ps -aux | grep condor
```

Az öt folyamat a következő:

- `condor_master`
- `condor_collector`
- `condor_negotiator`
- `condor_startd`
- `condor_schedd`

A *pool* minden más gépén a következő folyamatoknak kell futni a helyes működéshez:

- `condor_master`
- `condor_startd`
- `condor_schedd`

Végül a *submit-only* gépeken csupán két folyamatra van szükség:

- `condor_master`
- `condor_schedd`

Ha mindezek után kiadjuk a `condor_status` parancsot, akkor a központi gépet már mint a *pool* egyik tagját fogjuk látni:

```
$CONDOR/bin/condor_status
Name OpSys Arch State Activity
➥LoadAv Mem ActvtyTime
➥Mycluster
LINUX INTEL Unclaimed Idle
➥0.115 3567 0+00:40:04
Machines Owner Claimed
➥Unclaimed Matched Preempting
INTEL/LINUX 1  0  0  1  0  0
Total 1  0  0  1  0  0
```

Ha pedig a klaszter egyéb gépein is elindítjuk a `condor_master` folyamatot, akkor néhány percen belül azok is mint a *pool* tagjai fognak már megjelenni.
(Ez általában körülbelül öt percet vesz igénybe.)

Folyamatok indítása az új klaszteren

A klaszter teszteléséhez először is hozzunk létre egy egyszer „Hello Condor” alkalmazást:

```
#include
int main()
{ printf("Hello World!\n");}
```

Fordítsuk le a programot a *GCC* segítségével, majd a keletkezett bináris állományt indítsuk el a *Condor* fennhatósága alatt. Ehhez meg kell írnunk egy indítófájlt (***submit*** fájl).
Az indítófájl egy olyan információcsomag, amelyben meg kell adnunk a *Condornak*, hogy a kérdéses programot miként kezelje.
Ez a fájl tartalmazza tehát, hogy a program honnan veszi a bemenetét, hova írja a kimenetét, illetve hogy miként jelezze, ha hiba keletkezett, hol tárolja a hibaüzeneteket.
Esetünkben az indítófájl tartalma a következőképpen fest:

```
Universe = Vanilla
Executable = hello
Output = hello.out
Input = hello.in
Error = hello.err
Log = hello.log
Queue
```

Az első, ***Universe*** nevű bejegyzés azt adja meg, milyen környezetben kell a ***Condornak*** futtatni a kérdéses feladatot. Két említésre méltó ilyen „univerzum” van. A hosszú futásidejű dolgokhoz a `Standard` nevezetű használatos. (Jelen esetben a „hosszú” futásidő heteket vagy hónapokat is jelenthet.) A `Standard` univerzumnak ugyanis számos, a cél szempontjából igen kellemes tulajdonsága van. Képes elmenteni a futó program állapotát, és szükség esetén a „mondat közepén” folytatni annak a futtatását, sőt ha egy gép működésében hiba keletkezik, akkor képes más helyre átköltöztetni a rajta futó feladatokat. Ez természetesen óriási segítség lehet bizonyos helyzetekben, de van egy hátulütője: a `Standard` univerzum használatához magának az alkalmazásnak is ***Condor*** kompatibilisnek kell lennie, vagyis bele kell fordítani bizonyos kiegészítő rutinokat. Erre pedig nyilván csak akkor van lehetőség, ha rendelkezésünkre áll a forráskód.
A másik, `Vanilla` nevű univerzumot a rövidebb lélegzetű feladatokhoz szokás használni, de természetesen a hosszú futásidejűekhez is alkalmazható akkor, ha a gépek működése kellően stabil. Az előny itt az, hogy egyáltalán nem kell módosítani a binárisokat.
Léteznek a ***Condor*** alatt más univerzumok is. Van például `PVM`, `MPI` és `Java` univerzum, amelyek értelemszerűen a kérdéses technológiákra támaszkodó alkalmazások futtatásához használhatók. Ezekkel kapcsolatban a ***Condor*** dokumentációjában találunk bővebb információt.
Esetünkben a végrehajtható állomány neve `hello` (a hagyományos „Hello Condor” program), az általunk használt univerzum pedig a `Vanilla`.
Az `Input`, `Output`, `Error` és `Log` direktívák azt állítják be, hogy a ***Condor*** mely fájlokat kapcsolja a futó program ***stdin***, ***stdout*** és ***stderr*** csatornáihoz.
Végezetül a `Queue` direktívában adhatjuk meg, hogy a program hány példányban futtatható.
Ha elkészültünk a futtatás körülményeit leíró fájllal (***submit file***), akkor magát a végrehajtást a

```
condor_submit hello.sub
```

paranccsal kezdeményezhetjük.
A futó folyamat állapotát a `condor_q` paranccsal ellenőrizhetjük, amely kiírja, hogy a végrehajtási sor hány programot tartalmaz éppen, mi ezeknek az azonosítója (***ID***), valamint hogy éppen futnak, vagy végrehajtásra várakoznak. A program ezen kívül néhány statisztikai adatot is közöl a folyamatokról.
Bár ebben a cikkben egyelőre csak a ***Condor*** telepítésének és üzembe helyezésének részleteit tárgyaltuk, a rendszernek természetesen számos a gyakorlatban használható funkciója van, amelyekről mindenféle oktatóanyagokat találhatunk az interneten. Az elsődleges információforrás természetesen a ***Condor*** felhasználói kézikönyve, amit a ➲ www.cs.wisc.edu/condor/manual webcímen találunk meg. Ezt olvasgatva célszerű különös figyelmet szentelni a Standard és a Java univerzumnak. Az előbbi lehetőséget ad a folyamatok időnkénti ellenőrzésére, míg a másikkal – nevének megfelelően – ***Java*** alkalmazásokat futtathatunk.

Szintén hasznos, ha a *Condor* már a bootfolyamat részeként elindítjuk, különösen, ha gyakran használjuk a gépeket „klaszter üzemmódban”. A dolognak ráadásul az a haszna is megvan, hogy így ha lekapcsolunk egy munkaállomást, miközben fut rajta egy *Condor* folyamat, akkor annak biztosan lesz hova átköltözni (persze attól függően, hogy a `Standrad` vagy a `Vanilla` univerzumban fut). Ez pedig nagy szabadságot kölcsönöz a rendszergazdának az adminisztrációval kapcsolatban.

Túl a klasztereken

A *Condor* nem csak arra jó, hogy segítségével klasztereket hozzunk létre közönséges asztali gépekből. A rendszer egyik kiegészítése azt teszi lehetővé, hogy számítási feladatok ne csak egy adott klaszter gépei között tudjanak „szabadon költözni”, hanem akár két klaszter között is. A *Condor* nevezéktanában ezt a szolgáltatás *flocking*-nak (kb. falkába tömörülés, fürtözés) hívják, a lényege pedig az, hogy ha abban a *pool*-ban, ahol a feladatot elindítottuk, éppen nem áll rendelkezésre a szükséges számítási kapacitás, akkor a job automatikusan megtalálja magának az utat egy másik klaszterbe. Ez pedig végső soron egészen érdekes konfigurációk megalkotását teszi lehetővé.
A legegyszerűbb ilyen *flocking* konfiguráció létrehozásához nincs másra szükség, mint a *condor_config* fájlban megadni néhány speciális változót. Tegyük fel például, hogy van két klaszterünk, A és B, és azt szeretnénk elérni, hogy az A klaszteren elindított feladatok szükség esetén a B klaszterre költözzenek át. Tegyük fel továbbá, hogy az A klaszter vezérlőközpontja az *a.condor.org* címen található, míg a B klaszetré a *b.condor.org*. Ezekkel a konfigurációs fájl megfelelő része a következőképpen fest:

```
FLOCK_TO = b.condor.org
FLOCK_COLLECTOR_HOSTS =
➥$(FLOCK_TO)
FLOCK_NEGOTIATOR_HOSTS =
➥$(FLOCK_TO)
```

A `FLOCK_TO` változóban akár több *pool*-t is megadhatunk vesszővel elválasztva a megfelelő központi vezérlők neveit. A másik két változó általában ugyanoda mutat, mint a `FLOCK_TO` változó tartalma. A B klaszter konfigurációs fájljaiban olyan értékeket kell megadnunk, hogy azok fölhatalmazzák az A klaszter folyamatait a B klaszter csomópontjain való futásra. A következő példa egy ilyen beállítást mutat. Itt a `FLOCK_TO` változóhoz meglehetősen hasonló nevű `FLOCK_FROM` paraméter szolgál azoknak a „baráti” klaszereknek a felsorolására, ahonnan végrehajtási kérések érkezhetnek.

```
FLOCK_FROM=a.condor.org
HOSTALLOW_WRITE_COLLECTOR =
➥$(HOSTALLOW_WRITE),
➥$(FLOCK_FROM)
HOSTALLOW_WRITE_STARTD =
➥$(HOSTALLOW_WRITE),
➥$(FLOCK_FROM)
HOSTALLOW_READ_COLLECTOR =
➥$(HOSTALLOW_READ),
➥$(FLOCK_FROM)
HOSTALLOW_READ_STARTD =
➥$(HOSTALLOW_READ),
➥$(FLOCK_FROM)
```

A fenti beállítások tehát engedélyezik az A klaszter gépeinek, hogy a B csomópontjain fussanak, de ugyanez visszafelé már nem működik. Ha ezt is meg akarjuk oldani, akkor teljesen hasonlóan kell eljárnunk, csak a változókat és a szerepeket kell mindenütt megcserélni. A B klaszteren tehát a `FLOCK_TO`, `FLOCK_COLLECTOR_HOSTS` és a `FLOCK_NEGOTIATOR_HOST` változókban az A pool központi vezérlőjének címét kell megadnunk, míg az A klaszeteren a `FLOCK_FROM` változóval engedélyeznünk kell a B-ről érkező folyamatok végrehajtását.
Ügyeljünk a `HOSTALLOW_WRITE` és a `HOSTLLOW_READ` változók beállítására, ezek ugyanis azt határozzák meg, hogy mely gépek csatlakozhatnak az adott *pool*-hoz, illetve melyek azok a csomópontok, amelyek a klaszter állapotáról információt kérhetnek le, de nem csatlakozhatnak hozzá.
A *Condor* egy kifejezetten hajlékony módszereket kínál a gépek és jogosultságaik megadására. A következő beállítással például lehetőségünk van arra, hogy csak egy adott alhálózathoz tartozó gépek számára biztosítsunk olvasási jogosultságot:

```
HOSTALLOW_READ=127.6.45.*
```

Condor-G

A *Condor* segítségével kialakított klaszterek egymáshoz kapcsolásának másik módja a rendszer grides képességeinek kiaknázása. A *Condor* e tekintetben a *Globus Toolkitre* (➲ www.globus.org) támaszkodik. Ez egy olyan nyílt forrású eszközkészlet, amellyel *Grid*-en futtatható rendszereket és alkalmazásokat készíthetünk. A csomagban megtaláljuk a megfelelő szoftveres infrastruktúrát a folyamatok hitelesítéséhez, a jogosultságok kezeléséhez valamint a távoli eljárások indításához és az ehhez szükséges adatátvitel megvalósításához. A *Condor-G* egy olyan kiegészítés a *Condor* alaprendszerhez, amelynek segítségével alkalmazásaink *Grid*-képesek lesznek, vagyis képessé válnak távoli, csak a *Grid* valamely pontján megtalálható erőforrások használatára is.
A `Condor-G` tulajdonképpen nem más, mint egy átjáró a *Condor pool*-ok és a *Grid* között. Ez az a program, amely egyaránt kezeli a futtatási sorokat, valamint mindazokat az erőforrásokat, amelyeket az elindított folyamatok használni fognak, legyenek azok akár egyetlen klaszter gépein, vagy bárhol másutt a *Griden*. Feladata, hogy a *Globus* mechanizmusait használva biztosítsa az erőforrások és folyamatok közötti kommunikációt, és szükség esetén a fájlok mindkét irányú átvitelét. Aki többet szeretne megtudni a *Condor-G* használatáról, lapozza fel a *Condor* korábban már említett kézikönyvét a megfelelő fejezetnél.
Egy a *Globus* segítségével végrehajtható feladata indítási állománya a következőképpen nézhet ki:

```
executable = mygridjob
globusscheduler =
➥grid.sample.net/jobmanager
input=mygridi.txt
universe = globus
output = mygridjob.out
log = mygridjob.log
queue
```

Amint látható, mindössze két eltérés van a *Griden* illetve a helyi klaszteren (*pool*) végrehajtandó feladatok leíróállománya között. Először is az az univerzum, amelyben a grides alkalmazás fut a *Globus* nevet viseli.

Ez az a beállítás, amelyből a *Condor* tudja, hogy az adott *jobot* a *Globus* segítségével kell elindítania és a *Griden* kell futtatnia. Ennek megfelelően meg kell adnunk a *Globus* ütemező (*Globus Job Manager*) címét is a `globusscheduler` nevű változóban. Ez az ütemező egy a távoli gépen futó folyamat, és az a feladata, hogy folyamatosan kövesse a Griden futó alkalmazások *I/O* műveleteit, általános állapotát, és kezelje azok indítását. A Griden futtatott számítások megfigyelését ugyanúgy a `condor_q` paranccsal végezhetjük, mint a helyi folyamatok esetében.

Összefoglalás

Összességében elmondhatjuk, hogy a *Condor* egyedülálló lehetőséget kínál arra, hogy meglevő számítástechnikai infrastruktúránk segítségével olyan feladatokat oldjunk meg, amelyek az egyes egyes gépek képességeit messze meghaladnák. Telepítése és használata egyaránt könnyű, segítségével szinte pillanatok alatt építhetünk klasztert. Az így kialakított rendszer ráadásul skálázható, hiszen a *Condor* segítségével nem csak újabb csomópontokat csatolhatunk egy adott *pool*-hoz, hanem több ilyen klasztert is összefoghatunk egy fürtbe, sőt a megfelelő kiegészítésekkel a rendszer még a Grid részeként is funkcionálhat. Ennek megfelelően talán nem meglepő, hogy a *Condort* immár számos párhuzamos feldolgozást igénylő projekt megvalósítása során használták. Az egyik legújabb ilyen sikeres esettanulmány a *Micron Technologies*-től származik. A *Micron* a világ egy legnagyobb félvezető elemek gyártásával foglalkozó vállalata. 2006 áprilisában egy a *GridToday*-ben megjelent, a vállalat egyik vezető munkatársával készült interjú szerint nemrég megépítettek egy olyan rendszert, amely 11 *Condor* klaszterből, és összesen 11.000 processzorból áll. A gépek ráadásul a cég 11 telephelyén, négy különböző országban találhatók. A kérdésre, hogy miért éppen a *Condort* választották az illető azt válaszolta, hogy ennek számos oka volt, de elsősorban azért, mert a *Condor* minden, a vállalat számára érdekes platformot támogatott, széles körben használt, s így rendelkezésre álltak a szükséges információk az előzetes tervezéshez, jó a támogatottsága, no és persze mert nyílt forrású. A *Condor* létrehozott párhuzamos számítási kapacitás mára a *Micron* egyik igen komoly és értékes eszközét jelenti, amit a gyártásban, a tervezésben, a szoftverfejlesztésben, a biztonsági rendszer működtetésében, de még a kimutatások készítésében is felhasználnak. Elmondható tehát, hogy a *Condor* nem egyszerűen egy kísérleti eszköz, hanem olyasmi, ami a való életben is kiválóan megállja a helyét.

Irfan Habib

Egyetemi hallgató Pakisztánban a Nemzeti Műszaki és Tudományegyetem szoftvermérnöki karán. Évek óta komolyan érdeklődik a szabad és nyílt forrású technológiák iránt, kutatási munkáját pedig az elosztott és hálózati számításokkal kapcsolatos területen folytatja. Éppen innen ered a Condorral kapcsolatos érdeklődése is, hiszen ez mindkét említett területtel kapcsolatban áll. Irfant az irfan.habib@niit.edu.pk címen érhetjük el.