PVM és Condor, PVM-GRID

Párhuzamos és Elosztott rendszerek II.

Dr. Mileff Péter

Programok futtatása helyi PVM-démon felhasználásával

- A PVM egy olyan infrastruktúra, amely biztosítja a különböző gépeken futó processzek számára a kommunikációs csatornát.
- Definiál egy hozzávaló protokollt.
- Ahhoz, hogy ez az infrastruktúra egy program számára rendelkezésre álljon:
  - a program tulajdonosának a futtatás előtt létre kell hoznia a virtuális gépet,
  - azon rendszerek lehetnek tagjai, amelyeken a felhasználónak joga van használni a helyi PVM démont.
  - Ezek után elindítható a program, amely a virtuális gépen belül létrehozza a kitűzött feladatot megoldó PVM-processzeket.

2

Programok futtatása helyi PVM-démon felhasználásával

- A futtatás előtt az alábbi két dolgot kell tenni:
  - 1. A programot minden olyan architektúrán lefordítani, amely a virtuális gépet alkotó számítógépek között megtalálható.
  - 2. A lefordított állományokat abba a könyvtárba másolni, ahol a PVM a futás során a létrehozandó PVM-processzek forrását keresni fogja.
  - Ennek a jegyzéknek a neve a PVM dokumentációjában megtalálható. (Általában $HOME/pvm3/$ARCH/bin.)

3

Programok futtatása helyi PVM-démon felhasználásával

- A rendszeren belül a PVM-processzek tényleges futási helyét csak a PVM ismeri,
  - a programozó nem.
- A processzek megszületésükkor a PVM-rendszertől egyedi azonosítót kapnak,
  - melyek ismeretében egymásnak üzeneteket küldhetnek.
- A PVM gondoskodik arról, hogy egy üzenet eljusson a megadott azonosítóval rendelkező processzhez,
  - egy üzenetet küldő processznek csak a címzett processz azonosítóját kell ismernie, a futási helyét nem.

4

Programok futtatása helyi PVM-démon felhasználásával

- A PVM ezen a szolgáltatásokon kívül semmi többletet nem ad a futtatás megkönnyítésére,
  - különösen a hosszú ideig (napokig vagy akár hetekig) futó programok esetén jelent nehézséget.
- Nem ritka az ilyen hosszú futású program,
  - a PVM-et gyakran használják olyan mérnöki, fizikai, biológiai számításokat igénylő feladatokhoz, melyek során aránylag nem túl bonyolult műveleteket kell végrehajtani nagy mennyiségű adaton.
  - Az ilyen, és ehhez hasonló feladatok során elvégzendő lebegőpontos számítások pedig rendkívül megnövelik a futási időt.

5

Programok futtatása helyi PVM-démon felhasználásával

- A futtatás során a probléma jelentkezhet akkor, ha egyszerre több programot indítunk ugyanazon a virtuális gépen.
- Ilyenkor az ugyanabban a virtuális gépben futó, különböző alkalmazásokhoz tartozó PVM-processzek képesek egymásnak üzenetet küldeni,
  - nincs semmi védelem, ami elkülönítené őket.
  - Ez kiszámíthatatlan viselkedést okozhat a programokban, és ellehetetleníti az utólagos hibakeresést.

6

Programok futtatása helyi PVM-démon felhasználásával

- További nehézség:
  - a virtuális gép létrehozása (erőforrás-allokálás)
    - a rendelkezésre álló erőforrások közül ki kell választani azokat, amelyeket egy adott program futtatásához használni szeretnénk.
    - Amelyekből a virtuális gépet létre kívánjuk hozni.
  - PVM démonon keresztül végzett futtatás esetén ez teljes mértékben a program tulajdonosára hárul,
    - nem egyszerű feladat.
    - Tudnia kell hogy pontosan milyen gépek érhetők el a hálózatban, és nem árt ismerni azok aktuális leterheltségét sem.

7

Programok futtatása helyi PVM-démon felhasználásával

- A leterheltséget figyelembe véve:
  - elkerülhető, hogy egy túlterhelt gépen futó processz a kommunikációban fellépő késleltetésekkel visszafogja a többi, kevésbé terhelt gépen futót.
- Ilyen nem egyenletes leterheltség különösen ipari környezetben fordulhat el,
  - ahol a helyi klaszter munkaállomásait elsősorban a napi ügyek intézésére használják,
  - párhuzamos programok futtatása csak másodlagos.

8

Programok futtatása helyi PVM-démon felhasználásával

- Ilyen helyen csak olyan gépet ajánlatos a virtuális gép létrehozásakor felhasználni, amelyik éppen szabad:
  - vagyis annak tulajdonosa nem használ rajta semmilyen programot.
  - Hogy egy egyszerű felhasználó ezt hogyan tudja kideríteni, arra sok esetben egyáltalán nincs megoldás,
  - Ha mégis valamilyen módon képes felmérni a helyi hálózat gépeit, abból akkor sem tud következteti a következő pár perc vagy óra történéseire.

9

PVM PROGRAMOK FUTTATÁSA CONDORRAL...

10

PVM programok futtatása Condorral

- Az említett problémák megoldása:
  - szükség van egy olyan programra, amelyik egyrészt minden PVM-program számára új virtuális gépet hoz létre,
  - másrészt az aktuális terheltségtől függően változtatja a PVM-ek létrehozásában résztvevő gépek halmazát.
  - Ez a szoftver a felhasználóknak magasabb szintű szolgáltatást nyújt, mint egy sima PVM démon, elrejtve előlük a futtatás valódi menetét.
  - Ezek a szoftverkomponensek a **job menedzserek**.
  - Használata esetén a felhasználók nem a virtuális géppel, hanem vele tartják a kapcsolatot,
    - lényegében nem is kell tudniuk arról, hogy a PVM-programok futtatásához virtuális gépre is szükség van.

11

Condor

- Egy ilyen jobkezelő program a Wisconsin-Madison Egyetem által kifejlesztett **Condor**.
- A **cél**, amiért a Condort létrehozták:
  - lehetővé tenni az úgynevezett **High Throughput Computing**-ot (nagy áteresztő képességű rendszer - **HTC)**
    - nagyszámú, különböző tulajdonosokhoz tartozó számítógépek egy egységbe foglalásával.
    - Az olyan rendszereket, amelyek hosszú időn át sok számítást képesek elvégezni HTC rendszereknek nevezzük.
    - Tehát a Condor egy specializált feladatszétosztó rendszer számításigényes feladatokhoz.

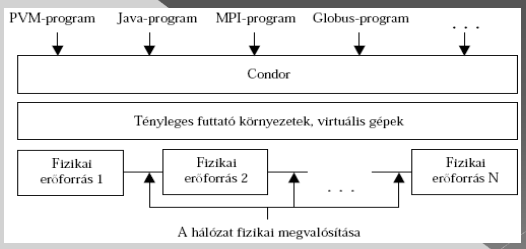
12

Condor

- Mint bármely más batch rendszer, a Condor is biztosít:
  - feladatsort (ide kerülnek be a végrehajtandó feladatok, job-ok),
  - ütemezési politikát, prioritási sémákat, erőforrások monitorozását és kezelését.
- Működése:
  - 1. A felhasználó odaadja a feladatot a Condornak, amely bekerül a feladat végrehajtási sorba.
  - 2. A meghatározott politika segítségével kiválasztja, hogy hol és mikor futtatja a job-ot, és miután a job lefutott, értesíti a felhasználót.

13

Condor rendszer sémája



14

Condor

- A Condor nem csak PVM-programokat, hanem más típusú, mind szekvenciális, mind párhuzamos programokat fogad.
- Elrejti a tényleges futtató környezeteket a felhasználók elől,
  - akiknek egy program futtatásakor csupán annyi a feladatuk, hogy elkészítenek egy megfelelő job-leíró fájlt,
  - melyet azután átadnak a jobkezelőnek.

15

Condor

- Condor segítségével kiépíthetünk egy grid stílusú számítási környezetet is:
  - „flocking" technológiája segítségével több Condor rendszert is összekapcsolhatunk.
  - Képes együtt működni számos grid-alapú számítási eljárással, protokollal.
  - Ilyen például a Condor-G, amely képes teljesen együttműködni a Globus kezelte erőforrásokkal.

16

A Condor

- **A Condor rendszer jellemzői:**
  - Elosztott, heterogén rendszerben működik,
  - Alapvetően a szabad CPU ciklusok kihasználására tervezték,
  - Képes egy működő feladatot áthelyezni az egyik gépről egy másikra (migráció),
  - Képes az ún. ***ClassAds*** mechanizmussal a rendszerben lévő változó erőforrásokat az igényeknek megfelelően elosztani.

17

A Condor

- A ***ClassAds*** lényege, hogy a rendszerben található erőforrások jellemzőkkel bírnak:
  - úgy mint teljesítmény, architektúra, operációs rendszer, bizalom, stb.
  - Egy job összeállításánál ezekre a jellemzőkre lehet igényeket előírni.
    - Amelyeket a Condor megpróbál kielégíteni.
    - A jellemzők között lehet preferenciákat is készíteni, amelyek akkor jutnak szerephez, ha több erőforrás is megfelel az igényeknek.

18

Condor

- Ebben a job-leíró fájlban szereplő alapvető adatok:
  - A futtatandó program típusa
  - A futtatható állomány neve
  - Futási paraméterek
  - Standard input-ot helyettesítő fájl neve
  - A program számára beállítandó környezeti változók
  - Futás közbeni eseményekről készítendő naplófájl neve

19

Condor

- A Condor a fájl tartalma alapján létrehoz egy új futási környezetet,
- elindítja a futtatható állományt a megadott környezeti változókkal,
- majd végig a futása során a fontosabb eseményekről bejegyzéseket készít a megadott naplófájlba.
- A program tulajdonosa a napló fájlból értesülhet például arról is, hogy job-jának a futása – akár sikeresen, akár sikertelenül – befejeződött.

20

Condor

- Fontos:
  - a Condor sok tekintetben magasabb szintű szolgáltatást nyújt, mint a PVM,
  - viszont cserébe fel kell áldozni a programok interaktivitását.
  - Condor-on keresztül ugyanis nem lehet interaktív alkalmazásokat futtatni.
  - Mint az előbbi felsorolásból látszik, a felhasználónak már a program elindítása előtt tudnia kell, hogy mi lesz annak bemenete,
  - ugyanis az adatokat a futtatás előtt egy állományba kell írnia, majd az állományt a leíró fájlban meghivatkoznia.

21

Condor

- A Condor garantálja:
  - hogy a futó program a hivatkozott fájlt fogja standard bemenetének tekinteni, és nem a billentyűzetet.
- Ez, a PVM-démonhoz képest jelentősnek tűnő megszorítás PVM-programok esetén szerencsére nem jelent gondot,
  - ugyanis a PVM alkalmazások által végzett számítások bemenő adatai általában előre ismertek, vagy futás közben generálódnak.

22

Condor, mint erőforrás menedzser

- A Condor tehát képes a felhasználó helyett a PVM létrehozására
  - és a futó programok folyamatos felügyeletére.
- Az ok azonban, amiért általában a felhasználók a Condor mellett döntenek:
  - az az erőforrás-menedzseri képessége.
- A Condor nem csak a futtatásra átadott programokat, hanem a hálózatban erőforrásként üzemelő gépeket is képes nyilvántartani.

23

Condor, mint erőforrás menedzser

- Az általa menedzselt hálózat minden gépén rendelkezik egy helyi démonnal
  - melyen keresztül folyamatosan figyelemmel kísérheti annak terheltségét.
  - A terheltség függvényében bármelyik erőforrást képes hozzácsatolni, illetve kivenni a programfuttatásra használható gépek halmazából.
- A leggyakoribb allokálási politika, amit Condor esetében alkalmaznak:
  - minden olyan gép, melyet x perce nem használt annak tulajdonosa, az kerüljön a központi Condor menedzser fennhatósága alá.

24

Condor, mint erőforrás menedzser

- Ha azonban megérkezik a tulajdonos, akkor őt illeti az elsőbbség.
- Ha az x időintervallumnak a hosszát a rendszergazda alkalmasan választja meg:
  - akkor egy ilyen megoldással kiszűrheti a korábban már említett kávészünetek miatt pihenő erőforrásokat a ténylegesen szabad gépek közül.
- A Condor egy program futtatásakor a politikája által szabadnak ítélt gépek halmazából választ a virtuális gép felépítéséhez.

25

Condor, mint erőforrás menedzser

- Az allokálási procedúra során nem csak az erőforrás-felajánlókra, de a job-ot futtatók igényeire is figyel.
  - A Condor felhasználók a job-leíró fájlokban megadhatnak a futtatáshoz használandó architektúrákra vonatkozó kikötéseket, prioritásokat.
- A Condor ezeket az információkat figyelembe véve:
  - megpróbálja a szabad gépek közül kiválasztani a legalkalmasabbakat a futtató környezet felépítéséhez.
  - Ha éppen nincs olyan architektúrájú gép, mint amit a kliens igényelt, akkor a Condor egyáltalán nem allokál erőforrást,
  - ennek okát pedig a naplófájlba írja.

26

Condor, mint erőforrás menedzser

- További probléma:
  - A Condor segítségével futtatott PVM programoknak más módon kell létrehozniuk a PVM-processzeket, mint PVM-démonnal való futtatás esetén.
  - Emiatt a felhasználóknak már a programjuk fejlesztése során tudniuk kell, hogy az később Condorral, vagy PVM-démonnal lesz-e futtatva.

27

A PVM és a Condor szerepe a Gridben

- A PVM összefogja a lokális hálózat gépeit, és belőlük nagyobb egységeket képez
  - Egy ilyen nagyobb egység = egy párhuzamos virtuális gép
  - a processzek számára biztosított kommunikációs infrastruktúra miatt sokkal nagyobb számítási kapacitást képvisel, mint az őt alkotó fizikai erőforrások kapacitásának az összege.
- A Gridben a cél a nagy kapacitást igénylő programok futtatása
  - Kézenfekvő a grid rendszer alapegységeit PVM –ekből létrehozni, nem pedig önálló gépekből.

28

PVM-Grid

- Az ilyen számítási rendszer neve **PVM-Grid**.
- **Jellemző tulajdonsága**:
  - A benne található erőforrások mind párhuzamos virtuális gépek.
- A PVM-Grid infrastruktúrát biztosít:
  - egyrészt a klienseknek a legmegfelelőbb PVM kiválasztásához,
  - másrészt a PVM processzeknek a kiválasztott virtuális gépen belüli hatékony kommunikációhoz.

29

PVM-Grid

- A helyi hálózatban élő PVM-démon nem biztosít elég magas szintű szolgáltatást,
  - vagyis a Condor-hoz kellett fordulni.
- A PVM-Grid esetén még ez a szint sem elegendő,
  - ugyanis a Condor-t általában csak a helyi hálózat erőforrás-menedzsereként használják.
  - Ilyen esetben csak a helyi hálózat gépeit használhatja programfuttatáshoz,
  - ilyenkor csak a helyi hálózat felhasználói érhetik el.
  - Ez utóbbi tulajdonság miatt egyetlen Condor-klaszter nem tekinthet gridrendszernek.

30

Barátságos pool

- A Condor lehetőséget biztosít **barátságos pool** létrehozására:
  - több klaszter összekötésével jön létre.
  - egyetlen Condor-példány az erőforrás- és a jobmenedzsere.
  - Ekkor lehetséges, hogy a **barátságos pool** egyik felhasználója által submitált job olyan gépeken is futni fog, melyhez az illetőnek nincs felhasználói fiókja.
    - Oka: a Condor megengedi a barátságos pool-on belül a jobok tetszőleges gépen való futását.

31

Barátságos pool

- **Felépítés két nagy hátránya:**
- 1. A klaszterek között meg kell nyitni minden egyes portot:
  - Ez akkora biztonsági rést jelenthet, hogy tényleg csak a legjobb „barátok" engedhetik meg maguknak a közös Condor használatát.

32

Barátságos pool

- **2. Konfigurációs fájlok:**
  - Ha valamelyik barátságos klaszter ki akar lépni a szövetségből, vagy egy új klaszter akar belépni a barátságos klaszterek közé,
    - akkor a pool minden egyes résztvevőjének módosítani kell néhány konfigurációs fájlt.
    - Ez néhány tag esetén még elviselhető, de nagyméretű Grid esetén nyilván kivitelezhetetlen.

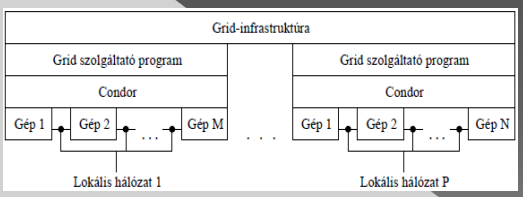
33

Barátságos pool

- Ha több száz, vagy ezer klaszterből álló Gridet akarunk létrehozni, akkor a Condor önmagában nem nyújthat megoldást.
  - Ennyi felhasználó nem bízik meg egymásban annyira, hogy barátságos pool-t hozzanak létre,
  - és egyébként is túl bonyolult a rendszer menedzselése.
- A Condor tehát PVM-gridekben a klaszterek közötti infrastruktúraként nem használható
  - viszont klaszteren belül jól funkcionál.
  - Emiatt érdemes a távoli kliensek számára futtató szolgáltatást nyújtó program, és a klasztert alkotó fizikai erőforrások közötti rétegként alkalmazni.

34

Condor



35

Condor

- Ilyen esetben:
  - a Grid infrastruktúrája a klaszterek közötti,
  - míg a Condor a klasztereken belüli erőforrás-allokációt végzi.
- A Condor a PVM-programok számára nem transzparens
  - a programoknak máshogyan kell Condor esetén a PVM-processzeket létrehozniuk.
  - ezért a Grid klienseinek tudnia kell a Condor létezéséről.
  - Ha viszont tudnak a Condor-ról, akkor akár a klaszteren belüli gépallokációhoz is kritériumokat adhatnak.

36

Condor

- Ha ezeket a kritériumokat a klaszter szolgáltató programja továbbítja a Condor-nak
  - akkor annak a gépallokáció során mind a rendszergazda által beállított politikára, mind a kapott igényekre tekintettel kell lennie.
- Ha viszont a kikötéseket a szolgáltató program nem továbbítja
  - akkor a Grid kliensei csak klasztert, és nem konkrét erőforrásokat választhatnak programjaiknak.

37

Köszönöm a figyelmet...

38