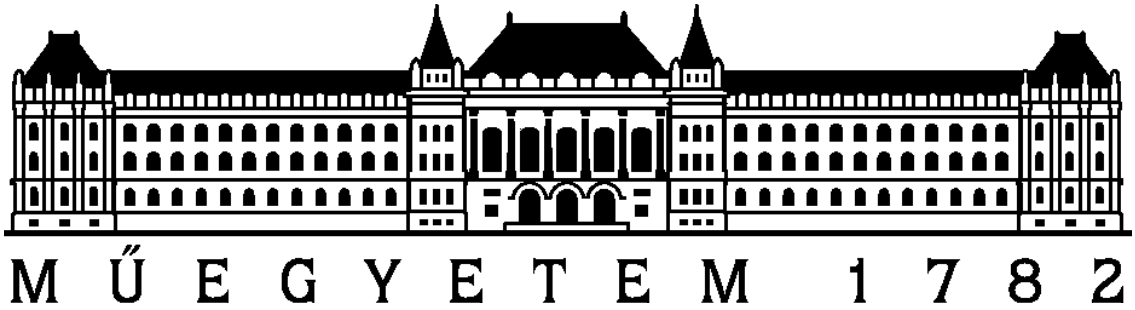


Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Diplomaterv

Medgyesi Zoltán

Konzulens: Micskei Zoltán
(Méréstechnika és Információs Rendszerek Tanszék)

2007

Nagy rendelkezésre állású kiszolgálófürtök vizsgálata

Diplomaterv-kiírás Medgyesi Zoltán műszaki informatika szakos hallgató részére

Ahogy az üzleti folyamatok egyre erőteljesebben támaszkodnak a számítástechnikai erőforrásokra, fokozódó igény mutatkozik a hibatűrő, a szükséges szolgáltatások folyamatos elérhetőségét biztosító, ún. fürtözött kiszolgáló rendszerekre. Míg korábban a fürtözésre inkább csak a nagygépeknél és a speciális célrendszereknél volt lehetőség, napjainkra minden jelentősebb operációsrendszer-gyártó kidolgozta a saját megoldását erre a területre, valamint a nyílt forrású rendszerekhez is készültek fejlesztések, így a fürtözés az alsó kategóriás, általános célú eszközökből álló számítógéprendszerekben is elérhetővé vált.

A fürtözési megoldások két nagy csoportját az elsősorban állapotmentes alkalmazások rendelkezésre állásának növelésére és horizontális skálázására alkalmas hálózati terheléselosztó megoldások és a főként állapottal rendelkező alkalmazások fürtözésére használható feladatátvételi megoldások alkotják. Bár a fürtözéssel kapcsolatos elméletek, alapvető protokollok több évtizedes múltra tekintenek vissza, a különféle gyártók és a nyílt forrású szoftverek fejlesztői többféle szemléletű fürtözési technikát is kidolgoztak, és jelenleg is folynak fejlesztések.

A diplomaterv célja ezeknek a technikáknak az áttekintése, a Windows Server operációs rendszerben elérhető megoldások részletesebb megismerése, valamint annak vizsgálata, hogy Petri-hálókkal hogyan lehet igazolni a fürtök kiépítésére fordított források megtérülését.

A diplomaterv kidolgozása a következő részfeladatok megoldását igényli:

- A hálózati terheléselosztó (load balancing) megoldások áttekintése, a Windows Server operációs rendszerben alkalmazott megoldás bemutatása.
- A feladatátvételi (failover) fürtök építésére alkalmas megoldások áttekintése, a Windows Server operációs rendszerben található implementáció bemutatása.
- Mintarendszer kiépítése a hálózati terheléselosztó és feladatátvételi fürtök működésének és alkalmazhatóságának szemléltetésére.
- Annak vizsgálata, hogy Petri-hálók segítségével hogyan modellezhetők a feladatátvételi fürtök, illetve hogyan igazolható, hogy a feladatátvételi fürtökkel fokozható a szolgáltatások rendelkezésre állása.

Budapest, 2007. január 30.

Micskei Zoltán
tanszéki konzulens

Nyilatkozat

Alulírott Medgyesi Zoltán, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Medgyesi Zoltán

Kivonat

Napjainkra gyakorlatilag minden üzleti folyamat számítógépekre alapul – a folyamatok egy része teljes mértékben elektronikus formát öltött, másoknál a hagyományos folyamatok hátttereként szolgálnak a számítógépes erőforrások, nyilvántartások. A számítógépek kiterjedt használata számos előnnyel jár, és újfajta lehetőségek, szolgáltatások előtt nyit utat, ám egyben a vállalatok függőségét, a technológiai hibáknak való kitettségét is fokozza.

Ebből a kitettségből fakadt az igény arra, hogy a számítógépekkel nyújtott szolgáltatások az idő minél nagyobb hányadában elérhetők legyenek a felhasználók számára, vagyis a lehető legmagasabb szintű rendelkezésre állást biztosítsák. A rendelkezésre állás fokozásának egyik módszere több független számítógép összekötése úgy, hogy az általuk nyújtott szolgáltatás egyetlen virtuális szolgáltatásként látsszon a felhasználók felé, és ennek a virtuális szolgáltatásnak a rendelkezésre állása nagyobb legyen, mint ami egyetlen számítógéppel elérhető lenne.

A dolgozatomban ezeket a fürtözési megoldásokat tekintem át. Dolgozatom első fejezetében összefoglalom, hogy pontosan milyen igényeket támasztunk a nagy rendelkezésre állású rendszerekkel kapcsolatban, és tágabb kontextusba helyezem a későbbiek során részletesebben is ismertetett technológiákat, a terheléselosztási és a feladatátvételi fürtöket. A második fejezetben bemuutatom a hálózati terheléselosztásra szolgáló módszerek és ezek lehetséges megvalósítás. A harmadik fejezetben a feladatátvételi fürtök koncepcióit, tipikus topológiáit és a fontosabb gyártók termékeit részletezem. Az elméleti részt gyakorlatiasabb témával, a korábban ismertetett technológiákat bemutató tesztkörnyezetek felépítésével folytatom, majd üzleti szempontból is megvizsgálom a fürtök építésével és a szolgáltatások rendelkezésre állásával kapcsolatos kérdéseket. A dolgozatomat az elmélethez visszatérve zárom, ennek során formális módszerekkel is bizonyítom, hogy fürtök használatával számottevő javulást lehet elérni a rendelkezésre állásban.

Tartalomjegyzék

1 Fürtözési technikák

1.1 Bevezető

Az elmúlt évtizedekben az emberiség egyre több feladatot bízott számítógépekre. Az eredetileg bonyolult számítások elvégzésére épített gépeket ma már sokkal inkább adatok tárolására, feldolgozására és továbbítására használjuk, az egyszerű nyilvántartások létesítésétől kezdve az elektronikus kapcsolattartáson keresztül egészen a pénzügyi műveletekig szinte minden számítógépen folyik. Az eleinte csak a vállalatok belső igényeit kielégítő kiszolgáló gépek mára az internethasználat, az integrált, egymásra épülő szolgáltatások, munkafolyamatok és az elektronikus kereskedelem terjedése nyomán külső személyek, az ügyfelek és az üzleti partnerek számára is szolgáltatnak információkat. Ezeknek a szolgáltatásoknak a folyamatos elérhetősége, rendelkezésre állása több szempontból is fontos, kiesésük szerződésszegést, ügyfélvesztést okozhat, ronthatja a vállalatról kialakított képet, továbbá rossz esetben, ha a szolgáltatás valamilyen munkafolyamat része, a leállás hatása továbbgyűrűzhet más szolgáltatásokra és rendszerekre is. A szolgáltatások egy része gyakorlatilag nélkülözhetetlen lett még a hétköznapi fogyasztók számára is, ilyen például a bankkártyás vásárlások ellenőrzése; az elszámolórendszer leállása nem csupán kiesett bevételt jelenthet a kereskedők és a bankok számára, de készpénz hiányában tehetetlenül toporgó vásárlók kerülhetnek kellemetlen helyzetbe, például a benzinkutaknál.

A számítógépes adatfeldolgozás megjelenése után már a korai időkben felvetődött, hogy az egymástól függetlenül végrehajtható feladatok vagy részfeladatok elvégzését párhuzamosítani kellene, felgyorsítva a végeredmény előállítását. A párhuzamosítás gyakorlatilag minden számítási, feldolgozási feladatnál nagy szerephez jut, hiszen a mindenkori processzorok sebessége az aktuális igényekhez képest mérve csekély, igazán jelentős teljesítményt csak nagyszámú processzor párhuzamos üzemeltetésével és a feladatok szétosztásával érhetünk el – így épülnek fel a szuperszámítógépek is.

A minél nagyobb teljesítményt egyetlen dobozba, de legalábbis egyetlen helyiségbe sűrítő szuperszámítógépek mellett más megoldások is születtek a teljesítmény és a rendelkezésre állás fokozására. A hétköznapi felépítésű és teljesítményű számítógépek redundáns összekötésével, fürtök építésével először az 1970-es évek végén próbálkoztak [2]. A nagy rendelkezésre állás problémájára a nagy rendelkezésre állású (Highly Available, HA) fürtökkel igyekeztek választ adni, a feladatok szétosztását pedig különféle általános célú terheléselosztási (load balancing) fürtökkel, kiszolgálófarmokkal, illetve kifejezetten számítási célokat szolgáló számítási fürtökkel oldották meg.

A fürtök építésének lehetősége az első időkben még erősen függött a hálózati szabványoktól és a protokolloktól – továbbá ezek hiányától és fejlesztésétől. Az az óta eltelt közel 30 év alatt az egyes területekre számos elméleti és gyakorlati megoldás, protokoll, algoritmus született. A nagy gyártók mindegyike elkészítette a saját megvalósításait, és a nyílt forrású közösség is számos fejlesztést indított, így a mai

felhasználók a különféle eszközök rendkívül széles köréből válogathatnak, és – kellő anyagi ráfordítás mellett – gyakorlatilag minden problémára megtalálható a megfelelő megoldás.

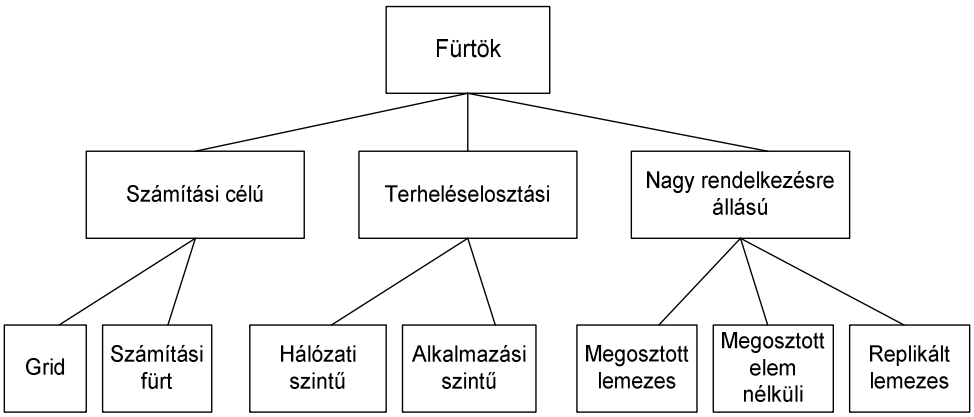
A nagy rendelkezésre állású kiszolgálófürtök és a szuperszámítógépek létezése és működése a hétköznapi felhasználók számára ma is jobbára rejtett, ám az elosztott számítási rendszerek a nagyközönség felé is láthatóvá váltak, amikor az 1990-es évek vége felé különféle közösségi számítási projektek indultak prímszámok keresésére, rákkutatásra vagy éppen a földön kívüli élet létezésének bizonyítására. Ezekbe a projektekbe a felhasználók ugyan csak mint a munka egy-egy apró szeletét elvégző ügyfelek kapcsolódhattak be a saját számítógépükkel, mégis alkalmasak voltak arra, hogy egyrészt a széles közösség számára is megmutassák az elosztott megoldásokban rejlő erőt, másrészt újabb fejlesztéseket ösztönöztek az elosztott rendszerek területén.

1.2 A fürt definíciója

A fürt definíciója nagyjából egységesen szerepel a szakirodalomban: különálló, hétköznapi jellemzőkkel rendelkező számítógépek – fürttagok – együttese, amelyek egymással együttműködve és azonos szolgáltatásokat, alkalmazásokat futtatva egyetlen rendszerként, virtuális kiszolgálóként jelennek meg az ügyfelek számára. A fürt tagjaival olyan hibatűrő, teljesítménynövelő megoldások – terheléselosztás, feladatátvétel – valósíthatók meg, amelyek egyetlen számítógép használatakor nem érhetők el.

1.3 Fürtözési megközelítések

A különböző problémákra számos eltérő ötlet és megoldás született, az alábbi ábrán ezek vázlatos felosztása látható. Az alábbiakban rövid kitekintést szeretnék adni a dolgozatom témáján kívül eső, egyéb fürtözési lehetőségekre is, továbbá segíteni szeretném az olvasót abban, hogy a későbbiekben ismertetett megoldásokat kontextusba tudja helyezni. Megemlíteném, hogy a fürtök kategorizálására más megoldások is születtek, ez csak egy a lehetséges felosztások közül.



1. ábra: A fürtözési megoldások csoportosítása

Röviden tekintsük át a fenti megoldásokat:

- A számítási célú (High Performance Computing, HPC) fürtök matematikai számítások, elemzések elvégzését teszik lehetővé. A feladatnak szétoszthatónak, párhuzamosan végrehajtható részekre bonthatónak kell lennie. Ilyen feladat például nagyszámú eltérő változat megvizsgálása.
    - A gridekben a fürt lazán csatolt. A fürtöt alkotó számítógépek egy központi számítógépről töltik le a feladatnak a rájuk eső részét, majd önállóan dolgoznak rajta, akár több hétig is. Amikor elkészültek, feltöltik a központba az eredményeket, és letöltik a következő adagot.
    - A számítási fürtök tagjai szorosabban csatoltak. A feladatok szétosztását itt is egy központi gép, a fürt „feje” vezérli. A tagok egymás között is továbbíthatnak üzeneteket, így egymás eredményeit vagy részeredményeit is át tudják venni saját feladatuk végrehajtása közben.
- A terheléselosztási fürtök célja az, hogy az ügyfelek kéréseinek kiszolgálását több olyan számítógép között osszák el, amelyek ugyanazt a kiszolgáló alkalmazást futtatják. Ezzel egyrészt könnyen lehet fokozni a rendelkezésre álló kiszolgálási kapacitást, másrészt tolerálni lehet egy-egy fürttag meghibásodását. A terheléselosztási fürtöknél alapesetben nem lehet előre tudni, hogy egy-egy kérés melyik fürttaghoz kerül, és adott ügyfél különböző kérései akár eltérő fürttagokhoz is juthatnak. A fürttagok mindegyike saját adattárolóval dolgozik; ha az egyik tag kiír a saját adattárolójára valamilyen változást, akkor arról a többi nem értesül. Ebből a két tulajdonságból fakad, hogy a terheléselosztási fürtökön jellemzően állapotmentes, tehát az ügyfélkapcsolatról a memóriában adatokat nem tároló, illetve írásvédett adatokkal dolgozó szolgáltatásokat szokás futtatni, például webkiszolgálót.
    - Az alkalmazási szintű elosztásnál a szolgáltatást több részre osztjuk, és az egyes komponenseit különböző gépeken futtatjuk. Ezzel egyrészt eloszthető a terhelés, másrészt több komponenspéldány telepítésével hibatűrés is megvalósítható. Hasonló eljárás az adatbázisok particionálása, például amikor az A-M kezdetű neveket az első, az M-Z kezdetű neveket pedig a második fürttag kezeli.
    - Hálózati szintű terheléselosztásnál a hálózati rétegben osztjuk el a kéréseket. Attól függően, hogy a hálózatnak az OSI referenciamodell szerinti melyik rétegében történik ez a művelet, hálózati, szállítási és alkalmazási rétegbeli hálózati terheléselosztásról beszélhetünk.
- A nagy rendelkezésre állású (HA) fürtök célja – mint a név is mutatja – a szolgáltatások elérhetőségének javítása. A HA fürtök alapkoncepciója az, hogy míg a szolgáltatást az egyik számítógép futtatja, addig legalább egy tartalék számítógép biztosítja a redundanciát arra az esetre, ha az aktív számítógép meghibásodna. Meghibásodás esetén egy tartalék veszi át a feladatot, miközben a szolgáltatás gyakorlatilag folyamatosan elérhető. A HA fürtök altípusairól a 3. fejezet elején lesz még szó.

A továbbiakban csak a hálózati terheléselosztási és a nagy rendelkezésre állású fürtökről lesz szó.

1.4 A fürtökkel szembeni elvárások

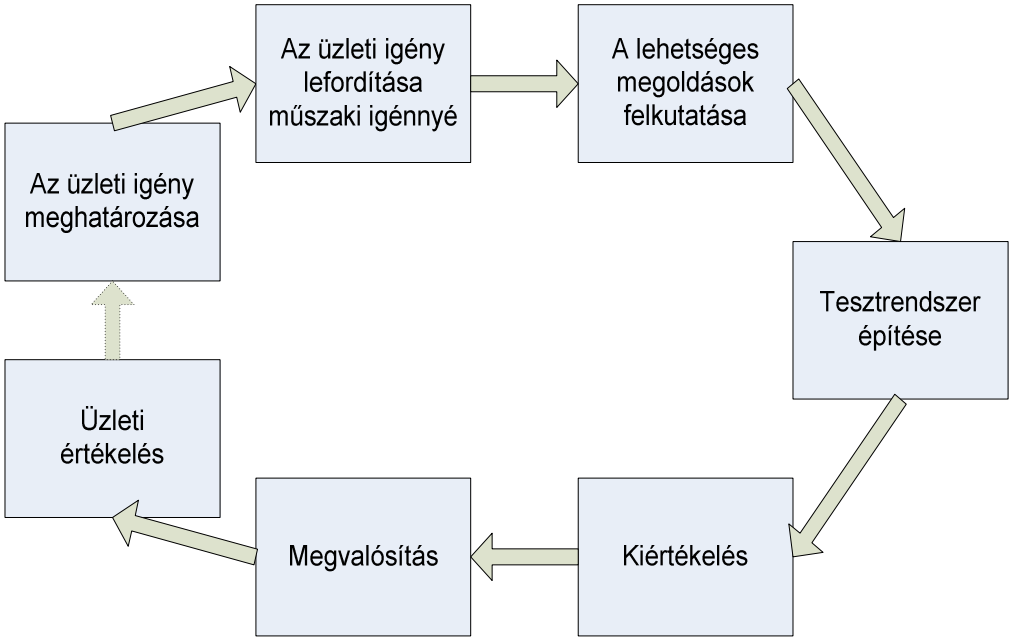
A fürtökkel – illetve a fürtök itt tárgyalt típusaival – szemben a következő általános elvárásokat fogalmazhatjuk meg [10]:

- Jelenlétük, működésük a lehető legnagyobb mértékben legyen észrevétlen a felhasználók számára.
- Beépített funkcióik révén legyenek alkalmasak bizonyos hibák elviselésére és a szolgáltatások fenntartására. Az alapvető elvárás az, hogy egy meghibásodás még ne okozza a szolgáltatás elérhetetlenné válását, általánosan pedig az, hogy a fürtözési megoldás legyen alkalmas a felhasználó által kitűzött rendelkezésre állás elérésére.
- Beépített funkcióikkal támogassák a hibák előjelzését, a bekövetkezett hibák gyors felismerését és a helyes működés gyors és automatikus helyreállítását.
- Akadályozzák meg az alkalmazások adatainak sérülését.
- Támogassák a bevált felügyeleti megoldások megvalósítását.

1.5 A fürttervezés folyamata

A korábbi pontok röviden összefoglalják, hogy milyen elvárások nyomán jöttek létre és mik is valójában a fürtök, valamint milyen fajtáik alakultak ki. Mint minden rendszer összeállítását, a fürtök felépítését is gondos tervezésnek kell megelőznie, hiszen kisebb részt hardver szempontjából, nagyobb részt üzleti szempontból, a szolgáltatások elérhetősége és az információk megőrzése miatt nagy értékű eszközökről van szó. (Lásd: 2. ábra)

Az üzleti igény meghatározása tulajdonképpen már megtörtént: valamilyen szolgáltatás folyamatos elérhetőségét biztosítani – mivel dolgozatom témáját az e problémára született megoldások képezik, feltételezem, hogy valóban erre van szükség, és nem az igény hibás megfogalmazása miatt tévedtünk erre a területre. A következő lépés ennek az igénynek a műszaki értelmezése, és annak felmérése, hogy az igény teljesítéséhez mely programok, eszközök folyamatos működése szükséges. Ezzel a lépéssel nem foglalkozom a dolgozatomban, feltételezem, hogy a tervezőnek sikerül kiválasztania azokat az alkalmazásokat, programokat, amelyeket folyamatosan kell működtetnie.



2. ábra: A tervezési folyamat lépései

Ha a műszaki igényeket sikerült körülírni, fel lehet kutatni a lehetséges megoldásokat. A dolgozatomban két nagy fejezetet szentelek a terheléselosztási és a feladatátvételi fürtöknek, mint széles körben elterjedt megoldásoknak.

Egy hagyományos fejlesztési folyamatban a következő lépés vélhetően a kiszemelt eszköz elméleti vizsgálata lenne. Az utóbbi években azonban nagy teret nyert és újfajta lehetőségeket nyitott meg a virtuális gépek használata. Úgy vélem, mivel alkalmazásukkal a szóba jöhető megoldások a leendő felhasználó egyéni igényei szerint, a saját alkalmazási és informatikai környezetében, a rá jellemző felhasználói szokásokat figyelembe véve értékelhetők, már csak olcsósága miatt is érdemes az elméleti vizsgálat előtt egy egyszerűbb tesztrendszert összeállítani. Erre mutat példát a 4. fejezet, amelyben a Windows Server 2003 operációs rendszerekkel épített tesztkörnyezetben szerzett tapasztalataimat összegzem.

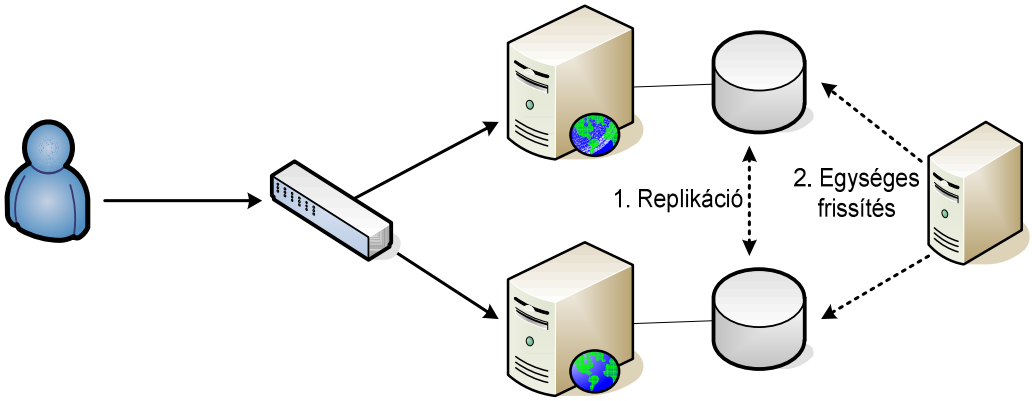
Ha a tesztrendszer alapján arra az eredményre jutunk, hogy a kiválasztott megoldás illeszkedik a meglévő környezetbe, akkor továbbléphetünk az elképzelt rendszer elméleti vizsgálata felé. A vizsgálat során kiderül, hogy egyrészt az első körben felvázolt rendszernek melyek a hiányosságai, másrészt vajon elérhetők-e vele a kitűzött célok. A 6. fejezetnek az ilyen vizsgálatok adják a tárgyát, míg az 5. fejezet a vizsgálatok megalapozásához és értékeléséhez próbál gyakorlatias támpontokat adni.

A tervezési folyamat utolsó lépése az üzleti szempontból történő értékelés, illetve – szükség esetén – a folyamat újrakezdése lehet. Itt évekre *visszatekintve* fogalmazhatók meg – szerencsés esetben – olyan kijelentések, mint például „a hibatűrő rendszernek köszönhetően x perc leállást előztünk meg, és ezzel y forint bevételkiesést akadályoztunk meg”. Ezzel a lépéssel nem foglalkozom, hiszen az egyes rendszereket mindig egyedileg érdemes utólagosan értékelni.

2 Hálózati terheléselosztás

2.1 Áttekintés

A hálózati terheléselosztás a terheléselosztási megoldások családján belül (lásd: 1.3, Fürtözési megközelítések) az ügyfelektől érkező kéréseknek a *hálózati rétegben* való szétosztását célzó megoldásokat öleli fel. Ha ki szeretnénk emelni a hálózati terheléselosztási – a továbbiakban röviden terheléselosztási – fürtök olyan jellegzetességeit, amelyek alapján tisztán elkülöníthetők a többi megoldástól, akkor egyrészt a hálózati rétegben történő terheléselosztást érdemes megemlítenünk, másrészt a fürttagok egymástól való függetlenségét. Ez a függetlenség egyrészt a független adattárolásban, másrészt az egymással való együttműködés hiányában mutatkozik meg. A függetlenség ugyan egyes megoldásoknál nem teljes, hiszen a fürttagok egymás között is kommunikálnak, ez a kommunikáció azonban az operációs rendszer vagy a hálózati alrendszer szintjén valósul meg, maguk a szolgáltatások egymástól függetlenül működnek.



3. ábra: Hálózati terheléselosztó fürt – áttekintő ábra

A fenti ábra egy két tagból álló, webkiszolgálókat tartalmazó terheléselosztó fürtre ad példát. A terheléselosztás módját itt még nem részleteztem, az ügyfél kérése egyszerűen egy hálózati kapcsolóba fut be, ahonnan valamelyik fürttaghoz kerül. A fürttagok saját adattárolóval rendelkeznek. A harmadik számítógép a háttérrendszer felől végzett felügyeleti és adatfrissítési célokat szolgálja.

2.2 Előnyök és hátrányok

A terheléselosztás jellegénél fogva alkalmas annak biztosítására, hogy a kéréseket kielégítő fürtöt könnyen, rugalmasan, a szolgáltatás kiesése nélkül lehessen bővíteni, illetve az igények visszaesése esetén a leépítés is hasonló módon végezhető el.

Mivel a fürt különálló számítógépekből épül fel, és a szolgáltatás fenntartásához akár egyetlen fürttag is elegendő lehet, a közös okra visszavezethető hibáktól eltekintve a terheléselosztó fürtök rendelkezésre állása rendkívül magas értéket is elérhet. A

rendelkezésre állás javításához kapcsolódik, hogy a fürttagokon futó operációs rendszer és alkalmazások hibajavításainak telepítése vagy verziófrissítése a szolgáltatás kiesése nélkül is végrehajtható, amennyiben a frissítést a kiszolgálókon egyenként, sorban végzik el. Köszönhetően annak, hogy a fürttagok egymástól függetlenül üzemelnek, az sem okozhat problémát, ha az egyes tagokon a frissítés elvégzésekor egy ideig az alkalmazások eltérő változata fut.

A hálózati terheléselosztó fürtökben a fürttagok jellemzően nem rendelkeznek közös adattároló eszközzel. Ennél fogva elsősorban statikus jellegű adatok szolgáltatására alkalmasak, például állandó adatokkal dolgozó webhelyek vagy FTP-helyek üzemeltetésére. Ha olyan szolgáltatást kell fenntartani, ahol a felhasználók írhatják is az adatokat, akkor két esetet különböztethetünk meg. Ha nem lényeges, hogy az adatok azonnal láthatókká váljanak a többi fürttagról is, akkor fájlrendszerek közötti időszakos szinkronizálással vagy replikálással (lásd: 3. ábra) megoldható a változások átvezetése. Ha az adatoknak azonnal láthatókká kell válniuk, például konzisztens adatbázis-kezelés céljából, akkor a hálózati terheléselosztó fürtök helyett más megoldást kell keresni. Ügyelni kell arra is, hogy a kiszolgálók tartalmát egységesen kell frissíteni, így elkerülhető, hogy a különböző fürttagokhoz kerülő ügyfelek eltérő tartalmat lássanak.

A terheléselosztó fürtökben a fürtözött szolgáltatást nyújtó alkalmazások magától a fürttől teljesen függetlenül üzemelnek, a fürt létezéséről semmilyen formában nem értesülnek. Ez egyrészt előny, hiszen az alkalmazásokat nem kell felkészíteni a fürtkeretprogrammal való együttműködésre, másrészt hátrány, hiszen az együttműködés hiányában arra sincs lehetőség, hogy az alkalmazások, szolgáltatások működését, terhelését figyelemmel kövessük. Emiatt kétféle hibára is fel kell készülnünk:

- Ha a terheléselosztó megoldás teljesen független a fürttagoktól, akkor előfordulhat, hogy olyan fürttagnak is továbbít kéréseket, amely időközben meghibásodott. Ebben az esetben a rendszergazda feladata a hibás számítógép kivétele a fürtből.
- A terheléselosztó megoldás hiába figyeli az egyes fürttagok működőképességét, ha a rajtuk futó szolgáltatásokkal nem tud együttműködni. Előfordulhat, hogy egy fürttag még működőképes ugyan, a szolgáltatás azonban már meghibásodott, és a terheléselosztás úgy továbbítja a számítógépnek az ügyfelek kéréseit, hogy az valójában nem tudja fogadni azokat. Ebben az esetben is a rendszergazda beavatkozása szükséges, a terheléselosztó megoldások jellemzően nem képesek a meghibásodott szolgáltatások újraindítására.

A fentiekből kitűnik, hogy a terheléselosztásnál a magas rendelkezésre állás a kiszolgáló gépek sokszorozásából fakad, és nem ezek kifinomult együttműködéséből.

Bizonyos esetekben a hálózati terheléselosztás a biztonság fokozására is alkalmas lehet. Egyes megoldásoknál a külső felhasználók nem tudnak információkat szerezni a rendszer belső felépítéséről, ami megnehezíti a támadások kivitelezését. A hálózati terheléselosztó struktúrák egy részénél akár különböző kiszolgálószoftvert vagy operációs rendszert futtató kiszolgálók használata is lehetséges, amivel a nyilvánosságra kerülő, távolról is kihasználható sebezhetőségekkel szemben lehet eredményesen védekezni.

A terheléselosztást leginkább webkiszolgálók fürtözésére szokták használni. Ugyanakkor a webszolgáltatások terjedésével, azzal, hogy a HTTP protokollt nem csupán weboldalak továbbítására használják, hanem egész más jellegű szolgáltatások elérésére is, újfajta követelmények jelennek meg például a munkamenetek kezelése terén. Ezek teljesítése egyes esetekben problémát jelenthet, más megoldásoknál viszont megfelelő konfigurálással viszonylag könnyen megoldható.

2.3 Alapvető struktúrák

A hálózati terheléselosztó megoldásokat alapvetően háromféle csoportba sorolhatjuk:

- round-robin DNS alapú,
- teljesen elosztott,
- központi elemre épülő.

Az egyes megoldások gyökeresen eltérő szemléletet tükröznek, illetve ár, a megvalósítás többletterhelése, eszközigénye és bonyolultsága, méreteződés, a felügyelet bonyolultsága, az elosztás kifinomultsága és hibatűrés szempontjából egyaránt rendkívül eltérő jellemzőket mutatnak. Általánosan kijelenthető, hogy az egyenletesebb, kifinomultabb elosztást eredményező megoldások drágábbak és nagyobb többlet ráfordítást igényelnek. Alapvető kérdés, hogy érdemes-e a kifinomultabb megoldásokra forrásokat áldozni, vagy ugyanezeket az erőforrásokat a nyers erő elvét követve inkább a feldolgozási kapacitás bővítésére kell fordítani. A megfelelő megoldást minden esetben az adott – műszaki és gazdasági – környezet figyelembe vételével kell kiválasztani.

2.3.1 Round-robin alapú megoldások

2.3.1.1 Round-robin DNS

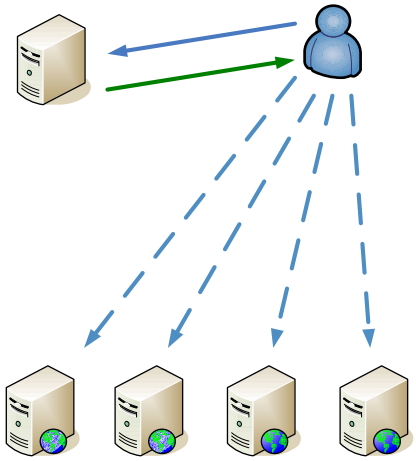
A round-robin DNS (RRDNS) több megoldás alapját is képezi, de önmagában is használható a bejövő kérések elosztására. A legősibb, ugyanakkor a legegyszerűbb és legszerényebb képességű terheléselosztó megoldás. Támogatása a legelterjedtebb DNS-kiszolgálóban, a BIND-ben is megtalálható [26], és például a Microsoft Windows Server rendszerekben is egyszerűen engedélyezhető.

Működése:

1. Az ügyfél a kérése elindításakor a DNS-kiszolgálóhoz intézi a kiszolgáló gépnevének feloldására vonatkozó kérdését.

2. A DNS-kiszolgáló válaszol az ügyfélnek. A kiszolgálón egy-egy gépnévhez több IP-cím is tartozhat, a megadott címek listáján a kiszolgáló végighalad, és minden egyes lekérdezésnél másik címet ad vissza.

3. Az ügyfél a kapott címtől függően mindig másik kiszolgálóhoz fordul.



4. ábra: A round-robin DNS működése

Előnyei:

- Egyszerű, kizárólag a DNS-kiszolgálón kell gondoskodni a megvalósításáról.
- A fürt tagjai egymástól teljesen függetlenek, akár különböző alhálózatokra is elhelyezhetők.
- A névkiszolgálók elhelyezésére vonatkozó szabályok (elsődleges és másodlagos kiszolgáló fenntartása, ezek külön alhálózatokra helyezése, független tápellátása) alapján az elosztást végző elem oldalán is magas rendelkezésre állás érhető el.

Hátrányai:

- Mivel az elosztást végző elem (DNS-kiszolgáló) és a fürttagok között nincs kapcsolat, a kiszolgálók kieséséről sincs jelzés. Ha egy fürttag meghibásodik, akkor a DNS-kiszolgáló továbbra is irányít hozzá ügyfeleket, egészen addig, amíg a rendszergazda be nem avatkozik, és a gépnévhez hozzárendelt IP-címek listájáról el nem távolítja a meghibásodott fürttagot.
- Külső tényezők, például a DNS-rekordok ügyféloldali gyorsítótárazása megzavarhatják. Hiába távolítja el például a rendszergazda a hibás fürttagot, ha az ügyfelek gyorsítótárában továbbra is szerepel az IP-címe.
- A kérések erőforrásigényét nem vizsgálja, így a gyakorlatban a fürttagokon rendkívül egyenlőtlen terheléseloszlás is kialakulhat.
- Több, egymástól függetlenül érkező kérésből álló munkameneteket nem képes kezelni, ugyanis nem biztosítható, hogy a kérések ugyanahhoz a fürttaghoz kerüljenek.

Az RRDNS továbbfejlesztett változatai:

- Súlyozott RRDNS: a gépnévhez rendelt IP-címekhez egyes esetekben súlyokat lehet rendelni, amelyekkel tükrözhető a fürttagok eltérő feldolgozási kapacitása.
- Alhálózattól függő RRDNS: a DNS-kiszolgáló képes lehet arra, hogy megvizsgálja az ügyfél IP-címét, és hozzá a hálózati topológiában legközelebb eső fürttag IP-címét adja vissza. Ezzel a megoldással vállalati hálózaton belül a

gerinchálózat terhelését lehet csökkenteni, világméretű hálózat esetén pedig a földrajzilag legközelebb eső kiszolgálóhoz lehet irányítani az ügyfelet.

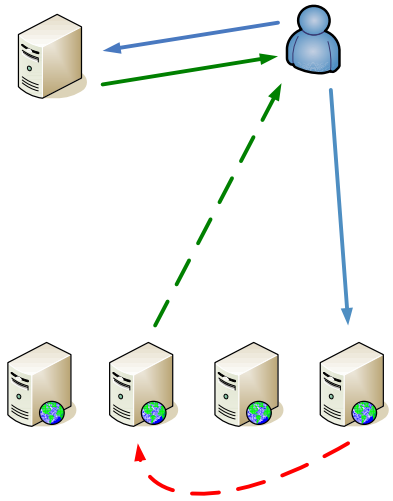
2.3.1.2 Distributed Packet Rewriting (DPR)

A Distributed Packet Rewrinting (elosztott csomagirányítás) módszere a round-robin DNS-re alapul, de kísérletet tesz arra, hogy az ügyfelek által küldött kérések átadásával egyenletesebb terhelést alakítson ki a fürttagok között. Ennek természetesen a megnövekedett bonyolultság az ára, ám köszönhetően annak, hogy az elosztás alacsony szinten, az IP protokoll rétegében történik, a ráfordítás megtérülhet.

Működése:

Az ügyfél az RRDNS megoldáshoz hasonlóan itt is bármelyik fürttag címét megkaphatja a DNS-kiszolgálótól, a különbség abban mutatkozik, hogy az egyes fürttagok át tudják adni egymásnak a kéréseket. Amennyiben erre sor kerül, úgy az átadást fogadó fürttag az átadó fürttag címét használva válaszol az ügyfélnek, aki az átadásról nem értesül, és továbbra is az eredeti kiszolgálóval kommunikál.

Az átadás kapcsán problémaként merül fel az átadott kérésekhez tartozó és a közvetlenül az ügyfelektől érkező csomagok megkülönböztetése, hiszen az előbbiekre az átadó fürttag címével, utóbbiakra a saját IP-címmel kell válaszolni. További probléma hogy az átadást fogadó fürttaggal közölni kell az ügyfél IP-címét. Mindkét kérdésre megoldást nyújt például az IP-IP beágyazás; ilyenkor az átadó fürttag módosítás nélkül átadja az átadást fogadó fürttagnak az ügyfél kéréscsomagját. Ezzel az átadást fogadó fürttag az átadó fürttag és az ügyfél IP-címéről egyaránt értesül, illetve a beágyazás ténye alapján az átadott csomagokat is meg tudja különböztetni [27].



5. ábra: Distributed Packet Rewriting

A kérések átadása többféle elven történhet:

- Állapotmentesen: ha az ügyfél kérését fogadó fürttag magasnak találja a saját terhelését, akkor átadja a kérést egy másik gépnek. Ilyenkor az átadást fogadó gép terhelését nem tudjuk figyelembe venni, az eljárás tehát egy-egy fürttag túlterhelését is okozhatja.

- Véletlenszerűen: az ügyfél kérését fogadó fürttag véletlenszerűen vagy továbbadja a kérést másik fürttagnak, vagy nem.
- Terhelésfüggően: a fürttagok nyomon követik egymás terhelését, illetve rendszeresen átadják egymásnak a saját terhelési értéküket. Ebben az esetben átadásra csak akkor kerül sor, ha az ügyfél kérését fogadó fürttag terhelése meghaladja a megadott küszöbértéket. Az átadás céljaként a legkisebb terhelésű fürttagot lehet kiválasztani. A terhelés meghatározásához például a memória-, processzor- és lemezhasználatot, valamint a megnyitott TCP-kapcsolatok számát lehet felhasználni.

Előnyei:

- Viszonylag egyszerű, mégis kifinomulttá tehető megoldás.
- Amennyiben a terhelési adatok továbbítása megoldott, akkor a fürttagok különböző alhálózatokon is elhelyezhetők.

Hátrányai:

- A fürttagokon kiegészítő szoftvert igényel.
- Az elosztás finomításához szükséges adatok összegyűjtése, küldése és fogadása többletterheléssel jár.

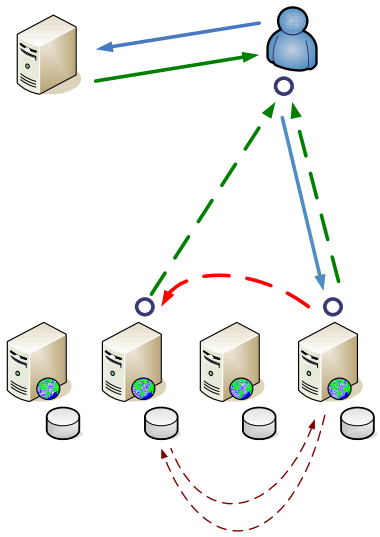
2.3.1.3 Socket Cloning (SC)

A Socket Cloning (socketek klónozása) szintén round-robin DNS alapú megoldás, hasonló a DPR-hez, azonban esetében a fürttagok megnyitott TCP-socketeket adnak át egymásnak.

Működése:

Az ügyfél a DNS-kiszolgálótól bármelyik fürttag IP-címét megkaphatja. A kiválasztott fürttag fogadja az ügyfél kérését, és az e célból létrehozott TCP-socketét mindvégig fenn is tartja, de szükség esetén más fürttagokon is létrehozza a másolatát. A klónsocketek közvetlenül az ügyfél felé adják át a választ, illetve a válasz – például egy weblap – egyes részeit [28].

Az SC hatékonysága elosztott gyorsítótárazás alkalmazásakor mutatkozik meg. Ha befut egy kérés valamelyik fürttaghoz, akkor annak a gyorsítótárában nem feltétlenül találhatók meg a kérés kiszolgálásához szükséges objektumok. A bevált megoldás ilyenkor a lassú lemezhasználat elkerülésére az objektumok átmásolása egy olyan fürttag gyorsítótárából, amely rendelkezik a szükséges példányokkal. Socket cloning alkalmazásával azonban elkerülhető, hogy az ügyfél kérését fogadó fürttag a belső hálózaton keresztül kénytelen legyen áttölteni a saját memóriájába az összes szükséges objektumot; helyette elegendő, ha klónsocketeket hoz létre a megfelelő fürttagokon, amelyek ezt követően közvetlenül az ügyfélnek küldik el az objektumokat. Példaként a weblapok átadása hozható fel, ezek általában több részből épülnek fel, a szöveg mellett képeket vagy egyéb beágyazott objektumokat is tartalmaznak, és ezeket egymástól függetlenül, akár párhuzamosan is át lehet adni az ügyfélnek.



6. ábra: Socket cloning elosztott gyorsítótárazással

Az SC megvalósításához, valamint a fürttagok és az ügyfél részvételével létrejövő útválasztási háromszög kezeléséhez viszonylag bonyolult szoftverre van szükség, a klónsocketek és forgalmuk kezeléséhez külön SC kiszolgálót, ügyfelet és útválasztót kell telepíteni. Az eredeti socket és a klón állapotát szinkronban kell tartani, ami vagy explicit módon, vagy az ügyféltől érkező csomagok, a TCP-nyugtaszám és az ablakméret alapján, implicit módon végezhető el. A klónsocketek kimenő forgalma az eredeti socket tulajdonosa számára láthatatlan, ezért a socketek megszüntetésére is vagy explicit megoldást kell alkalmazni, vagy implicitet, például időzítőre alapulót; utóbbit alkalmazza a [28] által tárgyalt tesztmegvalósítás.

Előnyei:

- Hatékony, a klónsocket az eredetitől függetlenül válaszol az ügyfélnek. Egyszerre több klón is létrehozható, amelyek párhuzamosan dolgozva kiváló teljesítménnyel tudnak válaszolni az ügyfélnek.

Hátrányai:

- Külön szoftvert, összetett állapotkövetést igényel.

2.3.2 Elosztott megoldás

A Microsoft Windows Server termékek az eddigiektől eltérő jellegű megoldást alkalmaznak. Működéséhez nincs szükség DNS-kiszolgáló közreműködésére, a terheléselosztást a fürttagok teljesen önállóan, ugyanakkor egymással együttműködve, egymást figyelve végzik [29].

Működése:

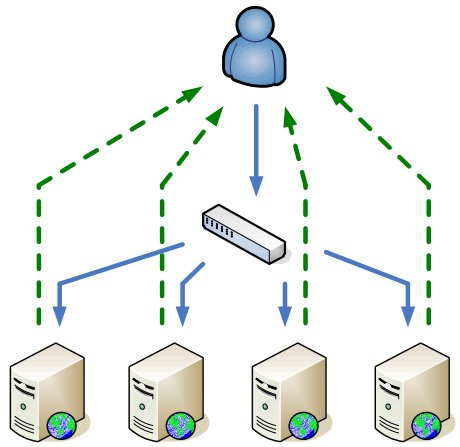
A fürt egy közös IP-cím alatt érhető el, erre a címre – a saját címe mellett – az összes fürttag válaszol. A hálózati hub vagy switch minden fürttaghoz eljuttatja a bejövő kéréseket. Minden fürttagon fut a hálózati protokollkészlet és a hálózati kártya illesztőprogramja közé beépülő Load Balancing szolgáltatás, és különféle adatok, többek közt az ügyfél címe alapján kiválasztja, hogy melyik fürttag szolgálja ki a kérést.

A kérést fogadó fürttagon a szűrőként üzemelő szolgáltatás továbbadja a kérést a felsőbb rétegek felé, a többi tagon viszont eldobja.

A fürttagok folyamatosan életjeleket továbbítanak egymás felé, így az egyes tagok kiesése rövid idő alatt észlelhető. A fürt felügyelete, bővítése, állapotának egységes állapotban tartása szintén a fürttagok között továbbított üzenetekkel folyik.

A beérkező kérések elosztása a forrás IP-cím és port alapján, állapotadatok alapján és részben véletlenszerűen történik – a pontos algoritmust nem publikálták.

A load balancing alapvetően viszonylag nagy mennyiségű, széles felhasználói körtől érkező, viszonylag kis méretű kérés esetén működik hatékonyan, például webszolgáltatásnál. Ha kicsi az ügyfélcímek tartománya, akkor az elosztás kevésbé jó. Az elosztás hatékony „beindulásához” legalább ötször annyi ügyfél szükséges, mint ahány tagot a fürt számlál.



7. ábra: A Windows Server termékek terheléselosztó megoldása

Előnyei:

- Egyszerű; nincs csomagirányítás, fürttagok közötti kapcsolatátadás, a szűrés sebessége eléri a 250 Mbit/s-ot.
- A kieső fürttagok felismerése gyakorlatilag azonnal megtörténik, a fürt újrakonfigurálása 10 másodpercen belül lezajlik.
- Nincs egyszeres hibapont.

Hátrányai:

- Méreteződése korlátos, 25 fürttag felett a teljesítmény romlik. A szolgáltatás legfeljebb 32 fürttagot támogat.

A Windows Server termékek Load Balancing szolgáltatásáról a 4. fejezetben lesz szó bővebben, ahol néhány további gyakorlatias technikai részlet is szerepel.

2.3.3 Központi elemet tartalmazó megoldások

Az előbbiektől ismét eltérő szemléletet tükröznek a központi elosztó elemre (dispatcher) épülő megoldások. Közös jellemzőjük, hogy egy elosztó elemre bízzák a kérések fogadását és a fürttagok közötti elosztását. Az elosztó elem figyeli azt az IP-címet,

amelyre az ügyfelek csatlakoznak; ez az IP-cím a virtuális cím, az ügyfélnek kizárólag ezt kell ismernie, a címre intézett kérések kiszolgálásának módjával nem kell foglalkoznia.

Az elosztó megvalósítása alapvetően kétféle módon történhet, a kapcsolatok átadásával (TCP handoff) vagy a kapcsolatok kettébontásával (TCP splicing). A piacon mindkét alapelv használatára találunk példát, a gyártók célkészülékeket és szoftveres megoldásokat egyaránt kínálnak.

Az elosztó elemek a 4. és a 7. rétegben is működhetnek, előbbi esetben útválasztó jellegű feladatokat látnak el, utóbbi esetben leginkább az alkalmazási rétegbeli proxykhoz hasonlíthatók.

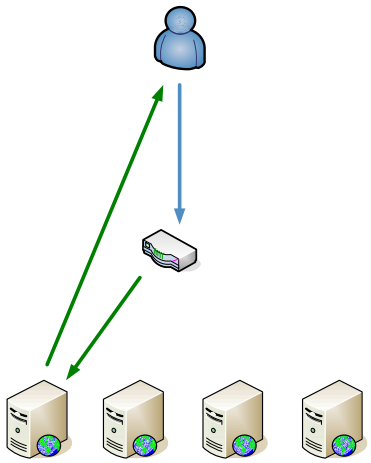
Az elosztóra épülő megoldások abból a szempontból rossz hatékonyságúnak tekinthetők, hogy a beérkező kérések feldolgozása két helyen – az elosztónál és a fürttagnál – is megtörténik.

További problémát jelent a munkamenetek, a perzisztens HTTP-kapcsolatok kezelése. A munkamenet elején ki kell választani a kéréssorozatot fogadó fürttagot, és a későbbiekben a választás – mivel az állapotadatoknak a fürttagok közötti átadására nincs mód – nem módosítható; csakhogy a kéréssorozat elejének vizsgálata nem feltétlenül szolgáltat információkat a későbbi kérések kiszolgálásából fakadó terhelésre nézve.

2.3.3.1 Kapcsolatátadás

Működése:

A központi elosztó elem fogadja az ügyfelek kéréseit, majd valamilyen szempontrendszer szerint továbbítja a fürttagok felé. Az elosztó elem a kérések végpontját az elosztás alkalmával ténylegesen továbbadja a tagoknak, a tagok a válaszukat már közvetlenül az ügyfélnek küldik.



8. ábra: A kapcsolatátadás működése

Előnyei:

- A válaszok kezelését a fürttagok végzik, továbbításuk már független a központi elemtől, és nem terheli azt.
- A fürttagok úgy látják a beérkező kéréseket, mintha közvetlenül kapták volna őket, így nincs szükség az alkalmazások módosítására.
- Lehetőség van arra, hogy a központi elem csak a megadott TCP-portokra beérkező kéréseket fogadja és továbbítsa a fürttagok felé, így egyfajta tűzfalként is védheti a fürttagokat.

Hátrányai:

- A központi elem egyszeres hibapont lehet.
- A fürttagok terheléséről kiegészítő megoldással kell információkat szerezni; az elosztás kifinomultsága ettől függ.

A kapcsolatátadás használata nem jellemző a kereskedelmi termékekre, azonban a Resonate cég (www.resonate.com) Central Dispatch termékével erre a megoldásra is találhatunk példát. Egyedisége miatt érdemes röviden áttekinteni a működését.

A Central Dispatch termékkel létrehozott terheléselosztási fürtökben kétféle fürttag lehet, ütemező és felügyelt tag. Az ütemező feladata az ügyfelek kéréseinek fogadása és továbbítása a felügyelt fürttagok felé, míg utóbbiak a tényleges kiszolgálást végzik. Ütemezőből a hibatűrés miatt lehet elsődleges és tartalék is, illetve kisebb forgalmú fürtnél többfunkciós fürttag is létesíthető, amely ütemezőként és felügyelt tagként is szolgál.

A fürt két lényeges eleme az RXP illesztőprogram és a Central Dispatch Agent, mindkét összetevőt mindegyik fürttagra célszerű telepíteni. Előbbi fogadja az ügyfélkapcsolatokat az ütemezőn, majd továbbítja őket a csomópontoknak, az utóbbinak pedig a fürttagok állapotának, terhelésének a követése és jelzése a feladata.

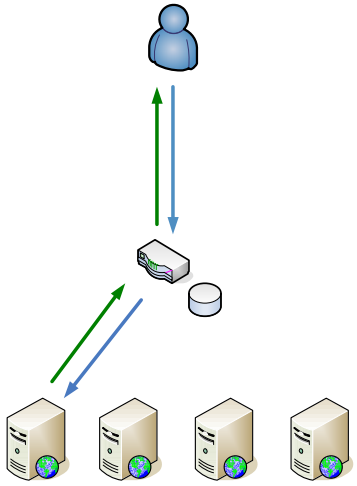
Ha a fürttagokon megtalálható az RXP illesztőprogram, akkor az ütemező ezen keresztül, a TCP Connection Hop szoftverösszetevő segítségével, TCP-alapú beágyazást alkalmazva továbbítja az ügyfelek kéréseit a tagoknak. A fürttagon megtörténik az eredeti ügyfélkapcsolat kibontása és helyreállítása, ezt kapják meg a kiszolgáló alkalmazások, amelyek közvetlenül az ügyfélnek adják át a válaszukat.

A Central Dispatch egy olyan működési módot is támogat, amelyben a két említett összetevő nem található meg a fürttagokon, az ilyen kiszolgálókat társult (affiliated) kiszolgálóknak nevezik. Ezeket további módokkal tudja kezelni, részben úgy, hogy az ügyfélforgalom teljes egészében – a válaszokkal együtt – az ütemezőn keresztül halad, részben úgy, hogy fennmarad a háromszög jellegű forgalmi minta.

A szoftver számos további szolgáltatást, kifinomult és sokféle szempont szerinti forgalomelosztást támogat, ám ezek részletes ismertetése nem célom. A cég webhelyén részletes dokumentáció található, továbbá a termék próbaváltozata is letölthető.

2.3.3.2 A kapcsolatok kettébontása

A kapcsolatok kettébontása az a módszer, amelyet a kereskedelmi – akár hardveres, akár szoftveres – megoldások jellemzően alkalmaznak. Szoftveres megoldásra példa az Octagate cég (www.octagate.com) Octagate Switch terméke, hardveres eljárást pedig a Cisco termékeiben láthatunk.



9. ábra: A kapcsolatok kettébontása

Működése:

A központi elem fogadja az ügyfelek kéréseit, majd a nevükben új kérést indít a kiválasztott fürttag felé. Amikor a válasz megérkezik, közvetítőként viselkedve továbbítja azt az ügyfél felé. A kapcsolatátadáshoz képest fontos különbség, hogy egyrészt minden adat áthalad az elosztón, másrészt az ügyfél kapcsolatának végpontja mindvégig az elosztó marad.

Előnyei:

- Az elosztón központi gyorsítótárazás valósítható meg.
- Az elosztóval magas szintű szolgáltatásokat lehet biztosítani, például át lehet venni a fürttagoktól a titkosítás vagy a hitelesítés kezelésének terhét.

Hátrányai:

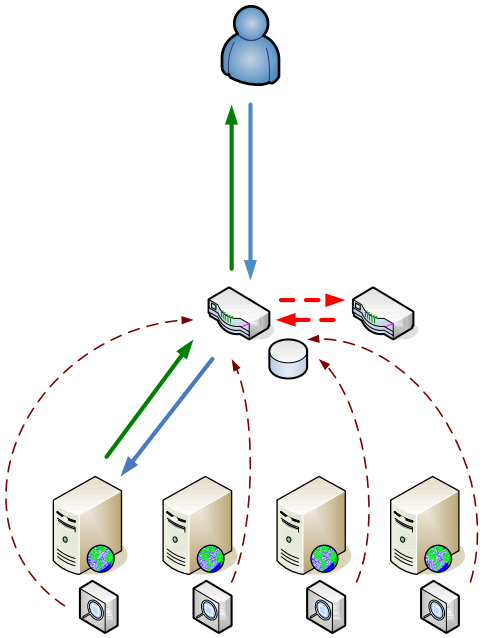
- Az elosztó egyszeres hibapont.
- Főleg 7. rétegbeli feldolgozásnál nagy mennyiségű kisméretű kéréssel az elosztó könnyen túlterhelhető, a fürt szűk keresztmetszetű pontjává válhat, illetve a feldolgozási hatékonyságnak a rendszer egészére nézve való romlását okozhatja.

2.3.4 Hibatűrő architektúra

A központi elosztó beépítésével egyszeres hibapontot vittünk a rendszerbe, holott a fürtözés egyik célja éppen az ilyen pontok meglétének elkerülése. Ezen a problémán úgy lehetünk úrrá, ha az elosztót is redundánssá tesszük, amivel a horizontális méretezést és a terheléselosztás gondolatát két szinten is megjelenítjük a rendszerben –

gyakorlatilag két fürtöt csatolunk egymás mögé. (10. ábra) A kétszintű fürtözést további megoldásokkal tehetjük kifinomultabbá.

Az egyik kiegészítő technika a socket cloning esetében már látott elosztott/együttműködő gyorsítótárazás. Az elosztók és a fürttagok egyaránt rendelkezhetnek gyorsítótárral, illetve a saját gyorsítótáruk mellett a többi fürttag gyorsítótárában található tartalom olvasására is képesek lehetnek – a tényleges hozzáférési lehetőségek és a hozzáférés sorrendje a megvalósítás függvényei. Az egyik lehetséges sorrend az, hogy a fürttagok az ügyfelek által kért objektumokat először a saját gyorsítótárukban keresik, majd a többi fürttag gyorsítótárában, és ha egyikben sem találják, akkor betöltik a helyi merevlemezről. Fontos kérdés, hogy mekkora objektumméretig érdemes alkalmazni ezt a technikát, hiszen a túl nagy objektumok hálózati átadása jelentős többletterhelést okozhat, ami szolgáltatásminőségi szempontból nem feltétlenül térül meg.



10. ábra: Példa hibatűrő architektúra felépítésére

Ha az elosztást kifinomulttá akarjuk tenni, akkor részletes információkat kell szereznünk a fürttagok terheléséről. Az információszerzés alapvető eszköze lehet az egyes fürttagokhoz továbbított kérések „nagyságának” követése, ám a hálózati forgalom alapján nem feltétlenül becsülhető meg, hogy a fürttag számára mekkora terhelést okoz egy-egy kérés kezelése. Részletes információkat ügynökök segítségével gyűjthetünk és adhatunk tovább az elosztó felé. Az ügynökökkel többféle rendszerparaméter is figyelhető, és az alapadatokból akár komplex terhelési mutatók is képezhetők.

A fentiekből látható, hogy az elosztás kifinomultsága különböző kiegészítő megoldásokkal viszonylag magas szintre emelhető, csakhogy a kifinomultság további feldolgozási kapacitást igényel. A tényleges megvalósításoknál mindig egyedileg kell megvizsgálni, hogy érdemes-e anyagi és műszaki erőforrásokat fordítani egy finomabb terheléselosztás üzembe helyezésére, vagy ugyanekkora ráfordítással inkább a tényleges

kiszolgálást végző hardveres erőforrásokat érdemes bővíteni. Az elosztó tehermentésítésére az alacsonyabb terhelésű fürttagokat is igénybe lehet venni, ám ebben az esetben is felmerül a kérdés, vajon az ehhez szükséges további szoftverelemeket érdemes-e kifejleszteni, megvásárolni és üzemeltetni.

A terheléselosztás tovább bonyolítható többszintű kiszolgálói rendszer létrehozásával. Építhető például olyan rendszer, ahol a párba kapcsolt terheléselosztó készülékek mögött fürtözött webkiszolgálók működnek, a webkiszolgálók pedig szintén fürtözött alkalmazáskiszolgálók szolgáltatásait veszik igénybe – kizárólag az alkalmazásarchitektúra korlátozza, hogy hány szinten jeleníthető meg a fürtözés.

3 Feladatátvételi fürtök

3.1 Bevezetés

A korábbi felosztásnál (1.3, Fürtözési megközelítések) már szerepelt a nagy rendelkezésre állású (HA) fürtök alapgondolata: aktív és tartalék számítógépek üzemeltetésével biztosítani a redundanciát és a szolgáltatások folyamatos futtatásának lehetőségét.

Kitérőként meg kell említeni, hogy ezeket a célokat más módszerekkel is el lehet érni. A DNS rendszernél például elsődleges és másodlagos kiszolgálókat használunk, az Active Directory címtárnál pedig – verziótól függően – elsődleges és másodlagos vagy egymással egyenrangú, az adatokat egymás között replikáló tartományvezérlők garantálják az adatok folyamatos elérhetőségét. Ezek a megoldások azonban nem fürtök, hiszen *nem egyetlen virtuális kiszolgálóként* jelennek meg az ügyfelek számára, így dolgozatomban nem részletezem őket.

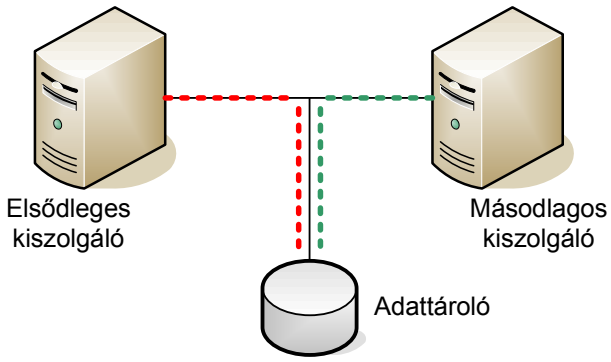
A HA fürtök jellegzetessége, hogy egyrészt az ügyfélkérések kiszolgálása során írt-olvasott adatokkal dolgozó szolgáltatást futtatnak, másrészt a szolgáltatás – éppen az adatok írása miatt – jellemzően csak egy aktív példányban fut. A HA fürtökön általában állapottal rendelkező alkalmazások futnak, amelyek az ügyfelek munkameneteihez kapcsolódó állapotadatokat tárolnak a memóriában. Természetesen nem lehet egyetlen általános definíciót kimondani, hiszen – ahogy a későbbiekben be fogom mutatni – az iparág szereplői által kidolgozott gyakorlati megoldások rendkívüli változékonyságot mutatnak.

Az adatok írási elérése, az állapotadatok kezelése, a feladatoknak a fürttagok közötti elosztása és a hibák kezelése számos érdekes problémát vetett fel; ezeknek a problémáknak a megoldására a HA fürtök területén belül is több megoldás született. Ugyan ebben a fejezetben csak a HA fürtök egy részéről, a feladatátvételi fürtökről lesz részletesebben szó, annak érdekében azonban, hogy tágabb kontextusba helyezhessük őket, érdemes elsőként a magasabb szintű modelleket áttekinteni [7].

3.2 Nagy rendelkezésre állás – általános modellek

3.2.1 Megosztott lemezes modell

A megosztott lemezes (shared disk) modellben az egyes fürttagok közös be- és kiviteli alrendszeren keresztül érik el a közös adattárolón lévő adatokat, és akár egyszerre is írhatják-olvashatják azokat. Az egyidejű hozzáférési lehetőségből fakadóan a szolgáltatások, legyen szó akár adatbázis-kezelésről, egyszerre több fürttagon is futtathatók. Az egyidejűség természetesen csak alkalmazási szinten biztosított, fizikai szinten – a konzisztencia megőrzése miatt is – gondoskodni kell a műveletek sorosításáról és a kizárólagos hozzáférésről. Ezeket a feladatokat a globális vagy elosztott zárolási szolgáltatás látja el.



11. ábra: A megosztott lemezes modell

A megosztott lemezes modellt alkalmazva elméletileg kiváló rendelkezésre állás érhető el, hiszen valamelyik fürttag meghibásodása semmilyen formában nem érinti a többi fürttag működését, a szolgáltatás futtatását egyetlen pillanatra sem kell megszakítani. Ugyanakkor a gyakorlatban több problémával is számolni kell, egyrészt meghibásodás esetén a hibás fürttagot meg kell akadályozni a közös adattárolóra vonatkozó zárolás fenntartásában és az adatok helytelen módosításában, másrészt hiba esetén újra kell konfigurálni a fürtöt, újra el kell osztani a beérkező kérések kezelését a tagok között.

A megosztott lemezes megoldás előnye, hogy további fürttagok hozzáadásával könnyen skálázható, hátránya ugyanakkor, hogy a közös adattároló könnyen szűk keresztmetszetté válhat. Mindig a tényleges alkalmazástól, elsősorban a lemezre történő írások és az olvasások mennyiségétől, arányától függ, hogy a modell mennyire állja meg a helyét.

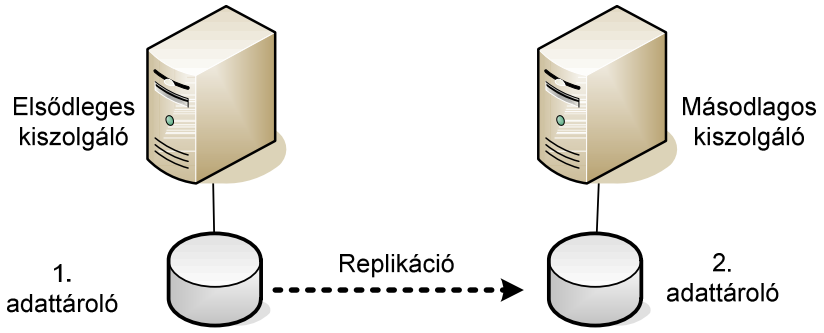
3.2.2 Megosztott elem nélküli modell

A megosztott elem nélküli (shared nothing) modellben egyszerre minden logikai és fizikai erőforrást kizárólag egy fürttag birtokolhat és kezelhet. A kizárólagos hozzáférés csak logikai szinten értendő, fizikai szinten mindegyik fürttagnak csatlakoznia kell például a közös adattárolóhoz, hiszen ennek hiányában az elsődleges fürttag meghibásodásakor a másodlagos tag nem tudná elérni az adatokat, és nem tudná futtatni a szolgáltatásokat. Mivel ebben az esetben nincs egyidejű többes hozzáférés, a lemezeléréshez nincs szükség globális zárolásra, ellenben az elérés kizárólagosságát ekkor is garantálni kell.

Egy fürt alapvető architektúrája sokszor ellentmondásosnak tűnhet. A Microsoft feladatátvételi fürtje például egy quorumnak nevezett erőforrással, egy egyszerű adatbázisban tárolja a fürt beállításait. A quorum az egyik fürtváltozatban közös SCSI-buszon elérhető merevlemezen, egyetlen példányban tárolódik, a másik fürtváltozatban viszont a saját, helyi merevlemezén minden fürttag rendelkezik egy-egy példánnyal. A két megvalósítás eltérőnek tűnhet, holott mindkét esetben csak egy fürttag birtokolhatja és kezelheti a quorum erőforrást, a modell megosztott elem nélküli jellege tehát nem sérül.

3.2.3 Replikált lemezes modell

A replikált vagy tükrözött lemezes modellben jellemzően két fürttag szerepel, egy aktív és egy tartalék. Miközben az aktív tag kiszolgálja az ügyfeleket, a rendszer folyamatosan tükrözi az adatok módosításait a tartalék tagra. Ha az aktív tag meghibásodik, a tartalék bármely pillanatban naprakész adatokkal tudja átvenni a feladatokat.



12. ábra: A replikált lemezes modell

A replikáció többféle szinten és eszközzel is megvalósítható:

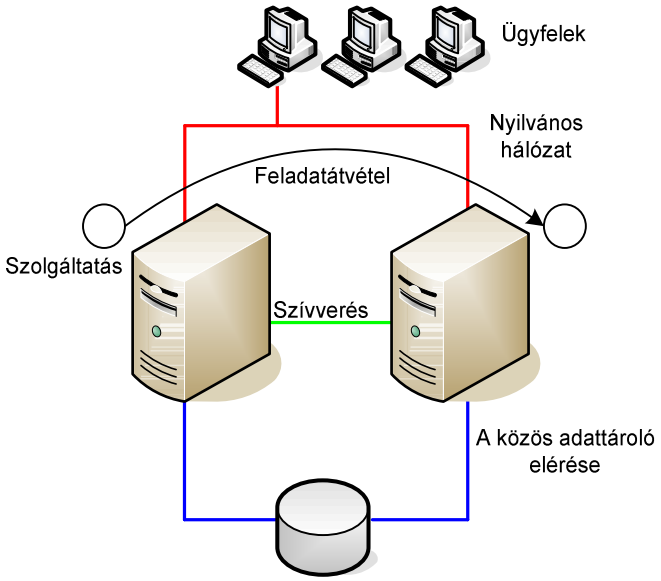
- Binárisan: A replikáció legalacsonyabb szintű módja a lemezre került adatok bináris továbbítása a tartalék adattárolóra. Ilyen megoldásokat a tárolórendszerekkel foglalkozó cégek, például az EMC kínálatában lehet találni.
- Fájl- vagy fájlrendszeri szinten: Ha a szolgáltatás fájlokkal dolgozik, akkor célszerű lehet fájlszinten replikálni az adatokat, ilyen szoftvert is több gyártótól lehet vásárolni.
- Integrált/alkalmazásszintű replikáció: Ha a szolgáltatás például adatbázis-kezelést végez, akkor a fájlszintű replikáció célszerűtlen, inkább a változásadatok továbbítására kell törekedni. Példaként az Oracle Streams említhető, amely adatfolyam formájában továbbítja a tartalék adatbázis felé az adatokat, tranzakciókat és egyéb eseményeket.

A replikációs eszköz a fürtszoftverrel is integrálható, illetve a két megoldás együttműködhet egymással; erre képesek például a Symantec Veritas termékcsaládjának megfelelő tagjai.

A replikált lemezes modell egyben arra is lehetőséget nyújt, hogy terheléselosztási fürttel állapotalapú alkalmazásokat védjünk a hibáktól. Ha kétirányú replikációval garantálni tudjuk az összes adatpéldány egységességét, akkor elvileg nem okoz problémát az adatok írása, ám a kölcsönös kizárásról, a zárolások kezeléséről, a műveletek sorrendjének megőrzéséről stb. ilyenkor is gondoskodni kell, amihez az alkalmazás részéről is megfelelő támogatásra lehet szükség.

3.3 A feladatátvételi fürtökkel kapcsolatos alapfogalmak

A jelen fejezet fő témáját jelentő feladatátvételi fürtök a megosztott elem nélküli modellt követik. További tárgyalásukhoz meg kell ismerni a velük kapcsolatos alapfogalmakat.



13. ábra: Alapszintű feladatátvételi fürt

A fenti ábrán egy alapszintű, két tagból álló fürt látható. A nagy rendelkezésre állású szolgáltatás alapesetben a bal oldali gépen fut, ez fogadja az ügyfelek kéréseit. Az adatok tárolása egy közös adattárolón történik, amely mindkét gép által elérhető SCSI merevlemez, iSCSI adattároló, SAN stb. lehet. A fürttagok *szívverések*, rövid hálózati üzenetek továbbításával jelzik egymásnak a működőképességüket. A fenti vonalak az ügyféloldali hozzáférést, a középső vonal a szívverések továbbítását szolgáló hálózatot jelöli, míg alul az adattároló elérésére használt hálózatot jeleztem.

Az ilyen fürtöknél általában fontos kikötés, hogy a fürttagok konfigurációja hardver és szoftver szempontjából azonos legyen; erre nemcsak a szolgáltatás zökkenőmentes futtatása, de a teljesítményméretezés miatt is szükség lehet.

Ha a bal oldali számítógép meghibásodik, akkor a fürtszoftver érzékeli ezt, a jobb oldali számítógépen elindítja a szolgáltatást, a bal oldali gépen pedig leállítja – ezt nevezzük *feladatátvételnek* (failover). Ettől kezdve a tartalék gép használja a közös adattárolót és fogadja az ügyfelek kéréseit.

Ha később a bal oldali gép ismét üzemképessé válik, akkor lehetőség van arra, hogy ismét ez futtassa a szolgáltatást. A feladatátvétellel ellentétes irányú műveletet *feladat-visszavételnek* (failback) nevezzük.

A legtöbb gyártó megkülönbözteti azt az esetet, amikor a szolgáltatások áttétele hiba miatt történik, és azt, amikor a rendszergazda kezdeményezi a műveletet, például azért, hogy valamelyik fürttagot ideiglenesen, például karbantartási célból kivehesse a fürtből. Az ilyen áttételeket *átkapcsolásnak* (switchover), egyes esetekben *felügyeleti*

feladatátvételnek (administrative failover), az ellenkező irányú műveletet pedig *visszakapcsolásnak* (switchback) nevezik.

A feladatátvétel és az átkapcsolás kapcsán fontos megemlíteni, hogy az előbbi esetében hiba történik, tehát nagyobb a valószínűsége az adatsérülésnek vagy az adatvesztésnek. Az átkapcsolásnál elméletileg minden rendszerelem kifogástalanul működik, ez a művelet tehát minimális kockázattal jár.

A 13. ábra ugyan csak egy szolgáltatást ábrázol, de a gyakorlatban nincs akadálya annak, hogy mindkét fürttagon fusson akár több szolgáltatás is, és a fürttagok kölcsönösen legyenek egymás tartalékai.

3.4 Alapelemek

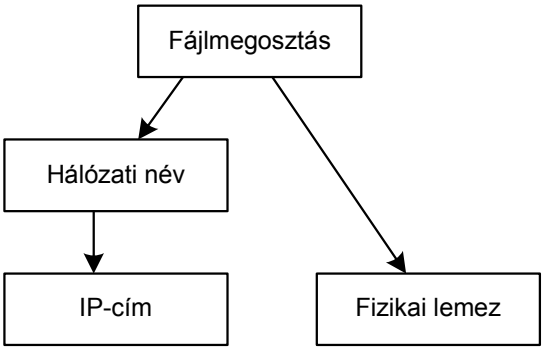
Az elmúlt évtizedekben jó néhány feladatátvételi fürtözési megoldás született, ám vannak olyan alapvető elemek és szolgáltatások, amelyek valamilyen formában – esetleg más névvel, más struktúrában – gyakorlatilag minden megvalósításban megtalálhatók, ezek alkotják a fürtök üzemeltetéséhez szükséges alapvető infrastruktúrát. Sok fejlesztő további szolgáltatások, lehetőségek beépítésével próbálta versenyképesebbé tenni a saját megoldását. Az alábbiakban először a minden fürtben megtalálható logikai elemek, a Microsoft megvalósítását példaként véve az alapvető szolgáltatások, majd a kiegészítő szolgáltatások vázlatos áttekintése következik.

3.4.1 Logikai egységek

A fürtök minden fizikai és logikai eszközt *erőforrásként* kezelnek. A fürt számára egyaránt erőforrás a fizikai lemez, a hálózati név, az IP-cím vagy a fürtözött szolgáltatás.

Az erőforrások *erőforráscsoportokat* alkotnak. A fürtszoftver feladatátvételkor és feladat-visszavételkor erőforráscsoportokkal dolgozik, ezzel biztosítva, hogy az egymástól függő erőforrások a megfelelő fürttagra kerüljenek.

Az erőforrások közötti függőségeket a *függőségi fával* ábrázolhatjuk. Egy erőforrás akkor függ a másiktól, ha az szükséges a működéséhez. Maga a függőségi fa általában ugyan nem jelenik meg a fürtszoftverben, de a függőségeket meg kell adnia a fürt üzemeltetőjének vagy a fürtözött szoftver fejlesztőjének. Függőség csak azonos erőforráscsoportban található erőforrások között lehet, hiszen az erőforráscsoportok egymástól függetlenül indíthatók, állíthatók le és helyezhetők át a fürttagok között.

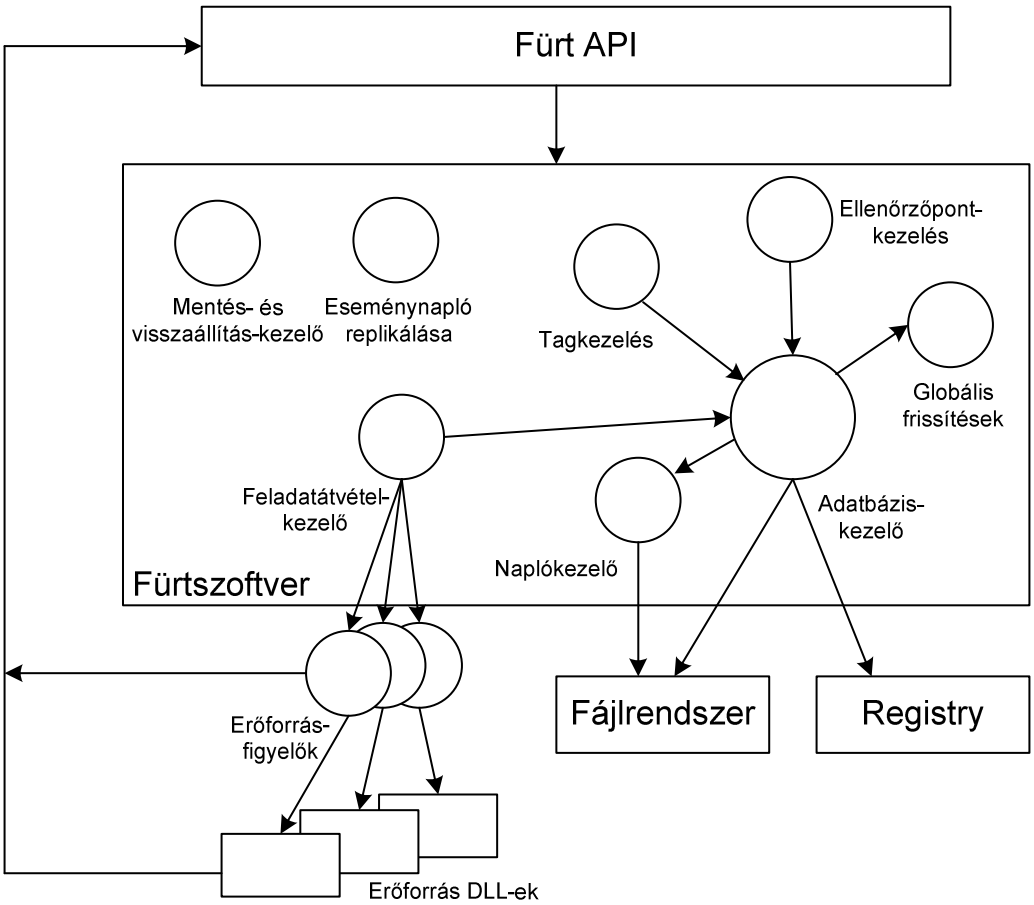


14. ábra: Példa függőségi fára

Feladatátvételkor a forrás fürttagon a fürtszoftver felülről lefelé haladva követi a függőségi fát, mindig azokat az erőforrásokat állítva le, amelyektől már nem függenek más erőforrások. A cél fürttagon fordított a folyamat, a fürtszoftver alulról haladva építi fel a fát, tehát minden erőforrást úgy indít el, hogy a működéséhez szükséges más erőforrások már készen állnak.

3.4.2 A fürtök építéséhez szükséges alapszolgáltatások

A Microsoft fürtözési megoldása az alábbi alapszolgáltatásokból épül fel [8].



15. ábra: A Microsoft feladatátvételi fürtszolgáltatásának felépítése

Az egyes szolgáltatások szerepe:

- **Tagkezelés** (node manager): A tagkezelő szolgáltatás mindegyik fürttagon fut, feladata az egyes tagok által a fürtről kialakított kép egységességének megőrzése. A tagkezelő szolgáltatás egyrészt rendszeres időközönként szívverés üzenetet küld a többi tag felé, ezzel jelezve saját működőképességét, másrészt figyeli a többi tag szívveréseit. A tagkezelő feladata a meghibásodott tagok eltávolítása a fürtből, valamint az újonnan csatlakozó tagok fogadása.
  A fürttagok kezelésében fontos szerepet játszik a tagságikép-kezelés is. Fő összetevője a csoportkezelő algoritmus, amelynek segítségével az egyes fürttagok kiesése vagy csatlakozása esetén a hibátlanul működő tagok mindegyikén egységes kép alakítható ki a fürt állapotáról.
- **Adatbázis-kezelő** (database manager): A fürt, az erőforrások, a tagság aktuális állapotát és a fürttel elérendő állapotot tárolja. Az összes tagon fut, az egyes példányok egységességét tranzakciókkal garantálja. A fürttel szorosabban integrált alkalmazások a saját adataik tárolására is használhatják a fürtszolgáltatás adatbázis-kezelőjét.
- **Ellenőrzőpont-kezelés** (checkpoint manager): Feladata a fontosabb változások rögzítése, így hiba esetén a fürt vissza tud lépni egy helyes állapotba.
- **Naplókezelő** (log manager): Az ellenőrzőpont-kezelővel együttműködve rögzíti a fürt konfigurációjának változásait. Ha egy fürttag meghibásodik, akkor a javításáig nem értesül a konfiguráció változásairól. Az ismételt üzembe helyezésekor a fürttag a naplókezelő segítségével, a változásokat végigkövetve hozza naprakész állapotba a fürtkonfiguráció nála lévő példányát.
- **Feladatátvétel-kezelő** (failover manager): Feladata egyrészt az, hogy az egyes erőforrások meghibásodása esetén megpróbálja újraindítani azokat, másrészt ha ez nem jár eredménnyel, akkor levezényelje a feladatátvételt. Az újraindítások és a feladatátvételek a vonatkozó házirendek, szabályok alapján történnek.
- **Globális frissítések** (global update manager): A szolgáltatás feladata a konfiguráció változtatásainak közlése a fürttagok felé úgy, hogy a változtatások időbeli sorrendje minden fürttagon azonos legyen, és a változások minden tagon egységesen végre legyenek hajtva. Ha egy fürttag változtatást akar végrehajtani, akkor először globális zárolást kell szereznie a konfigurációra. Amint ezt megkapta, kezdeményezi a módosítást, amely csak akkor jut érvényre, ha az összes tagnak sikerült fogadnia. A globálisfrissítés-kezelő szolgáltatásra támaszkodik többek közt az adatbázis-kezelő és a tagkezelő is.
- **Mentés- és visszaállítás-kezelő** (backup and restore manager): Feladata egy API biztosítása a külvilág felé, amelyen keresztül biztonsági másolat készíthető a fürtadatbázisról, valamint szükség esetén visszaállítható a fürtadatbázis korábbi állapota.
- **Eseménynapló replikálása** (event log replication manager): A szolgáltatás gondoskodik arról, hogy a fürttel kapcsolatos, a fürttagok által a helyi eseménynaplóba küldött bejegyzések a többi fürttagra is átkerüljenek; ezzel

garantálva az eseménynaplók egységességét és azt, hogy szükség esetén bármelyik tagon végig lehessen követni a naplóbejegyzéseket.

Ugyan nem részei a fürtszoftvernek, ám annak nélkülözhetetlen kiegészítői a következő elemek:

- **Erőforrás DLL-ek** (resource DLLs): A különböző típusú erőforrások kezelését erőforrás DLL-ek segítségével végzi a fürt. Feladatuk a szükséges absztrakciók biztosítása, valamint kommunikációs felületként és felügyeleti eszközként szolgálnak. A fürtszoftver eleve tartalmazza az általános erőforrások, például az IP-címek kezeléséhez szükséges DLL-eket, de a fürtözött működésre felkészített alkalmazásokhoz is tartozhatnak ilyenek, ezek az adott szoftver telepítésekor adódnak hozzá a rendszerhez.
- **Erőforrás-figyelők** (resource monitors): Feladatuk az egyes erőforrások és a fürtszoftver közötti kommunikáció biztosítása, illetve az erőforrások működésének figyelése. Más rendszerekben a figyelés szondákkal vagy parancsfájlokkal is történhet.
- **Felügyeleti felület**: A fürt felügyeletéhez szükséges kezelőfelület.

3.4.3 Kiegészítő szolgáltatások

Az alapszolgáltatások mellett az egyes megoldások fejlesztői számtalan további ötletet valósítottak meg annak érdekében, hogy megkönnyítsék a fürtök felügyeletét. Néhány ezek közül:

- A meghibásodott gép újraindítása: Egyes fürtszoftverek alkalmasak arra, hogy egy kiegészítő hardver segítségével automatikusan megszakítsák a meghibásodott fürttag áramellátását (STOMITH, Shoot The Other Machine In The Head, „lődd fejbe a másik gépet”, más forrásokban STONITH, a „machine” helyett „node”, csomópont szerepel). Így komolyabb szoftverhiba esetén is esély kínálkozik a fürt automatikus helyreállítására.
- Integrált szoftverfrissítés: A fürtöknél a szoftverek frissítését gördülő frissítéssel (lásd még: 3.6, Jellegzetes problémák a fürtökben) kell elvégezni, amelynek során egy-egy fürttagról feladatátvétellel eltávolítják a szolgáltatásokat, elvégzik a frissítést, majd újra üzembe állítják a fürttagot. A frissítési funkció az operációs rendszer hasonló szolgáltatásával is integrálható.
- Egyéb hardverek figyelése: A szívverésalapú figyelés egyéb eszközökre vagy a fürt hálózati kapcsolatát biztosító készülékre is kiterjeszthető.
- Kifinomult feladatátvételi házirendek: Több tagból álló és több szolgáltatást is futtató fürtöknél pontos szabályozásra lehet szükség annak meghatározásához, hogy mely szolgáltatások fussanak azonos fürttagon, melyek nem futhatnak ugyanazon a számítógépen stb. A házirendek egyéb elemekkel is bővíthetők, például eltérő napszakokban eltérő házirendek alkalmazhatók, a feladatátvételi feltételek felhasználó által definiálhatók vagy a rendelkezésre állás növelése érdekében pontos feladat-visszavételi feltételek adhatók meg.

- Felügyeleti integráció: Integráció a rendszerfelügyeleti szoftverekkel, a címtárakkal és a felügyeleti szabványok – CIM, SNMP – támogatása.

3.4.4 Fürtözés az alkalmazások szemszögéből

Fürtözés szempontjából az alkalmazások kétfelé oszthatók, a *fürttudatos* és a *nem fürttudatos* alkalmazásokra.

- A fürttudatos alkalmazások fel vannak készítve a fürtözött futtatásra, és a fürt API-n keresztül – például a hibák jelzésével – aktívan együttműködnek a fürttel. A fürttudatos alkalmazások a hardver- és a szoftverhibáktól egyaránt védhetők.
- A nem fürttudatos alkalmazások nincsenek felkészítve a fürtszoftverrel való együttműködésre. Ezeket az alkalmazásokat a fürtszoftverek általános alkalmazásként vagy szolgáltatásként kezelik. Mivel az ilyen alkalmazások állapotát a fürt nem tudja vizsgálni, jellemzően csak a hardverhibáktól védhetők; ha az adott alkalmazást futtató fürttag meghibásodik, a fürtszoftver egyszerűen elindítja a másik fürttagra telepített példányt.

Az alkalmazások fürtözésének további követelményei is lehetnek, például:

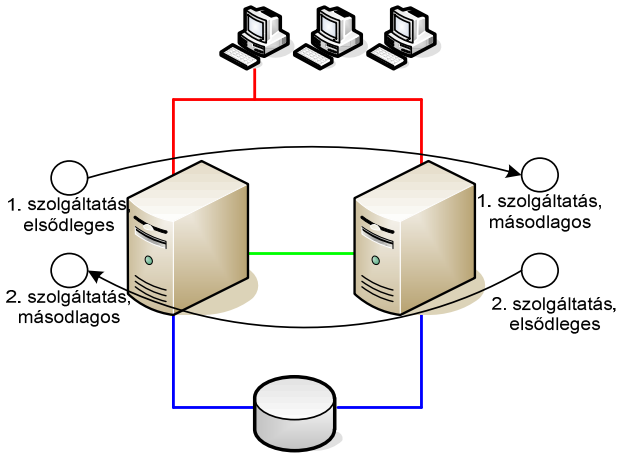
- Korlátozott lehet a használható hálózati protokollok köre. A Microsoft fürtözési megoldása például csak a TCP/IP protokollkészletet támogatja.
- A konfiguráció és az alkalmazásadatok tárolásának konfigurálhatónak kell lennie. Feladatátvétel esetén az alkalmazás beállításainak és adatainak elérhetőknek kell lenniük. Ha nincs lehetőség arra, hogy a beállítások és az adatok a minden fürttag által elérhető közös lemezen tárolódjanak, akkor az alkalmazás nem fürtözhető.
- Csökkenteni kell a memóriabeli állapotadatok mennyiségét. Minél több állapotadatot tárol az alkalmazás a memóriában, annál több információt veszít el feladatátvétel esetén.

3.5 Feladatátvételi topológiák

3.5.1 Áttekintés

Az előző szakaszban említett modellek közül – bár mindegyikre találunk megvalósítást – a megosztott elem nélküli a leginkább elterjedt. A megosztott elem nélküli modell jellegzetessége a feladatátvétel. A fürtök kiépítésekor az egyes számítógépek között többféle feladatátvételi topológia is kialakítható, ez határozza meg, hogy mely fürttagok szolgálnak egymás tartalékaiként, illetve az egyes fürttagok milyen szerepet (aktív kiszolgálói, tartalék) játszanak. Ki kell emelni, hogy a topológiák a fürtszoftverektől gyakorlatilag függetlenek, kialakításuk – a megfelelő feladatátvételi szabályok megadásával – a rendszergazda feladata [9].

3.5.2 Feladatátvételi pár



16. ábra: A feladatátvételi pár topológia

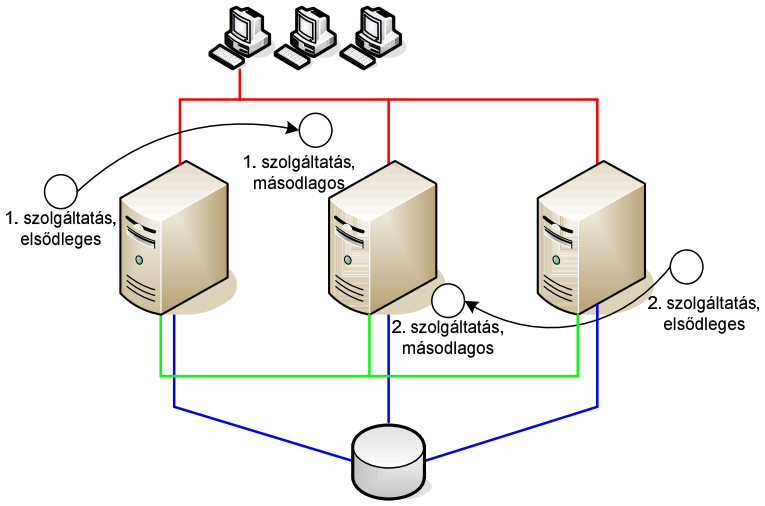
A feladatátvételi pár a legegyszerűbb topológia. Alapesetben egyetlen alkalmazást futtat az elsődleges kiszolgálón, ennek meghibásodása esetén a másodlagos kiszolgáló veszi át a feladatot, és vele együtt a mindkét tag által elérhető adattároló kezelését. Az alapkiépítés a különféle elemek megkettőzésével, például kettős ügyféloldali hozzáféréssel vagy szívverés-továbbítási kapcsolattal bővíthető.

Ha alapesetben csak az egyik fürttag futtat szolgáltatást, a másik pedig várakozik, akkor aktív-passzív, ha pedig mindkettő futtat szolgáltatást, akkor aktív-aktív topológiának is szokás nevezni a fenti elrendezést.

A feladatátvételi pár hátránya, hogy csak az egyik fürttag van kihasználva, a másik csak feladatátvétel esetén jut szerephez. Ezen úgy lehet javítani, hogy az ábrán látható módon mindkét fürttag elsődleges kiszolgálóként futtat egy-egy alkalmazást, és meghibásodás esetén a másik fürttagra történik a feladatátvétel. Ebben az esetben ügyelni kell arra, hogy mindkét fürttagnak elegendő kapacitással kell rendelkeznie mindkét szolgáltatás egyidejű fenntartásához.

3.5.3 Forró tartalék

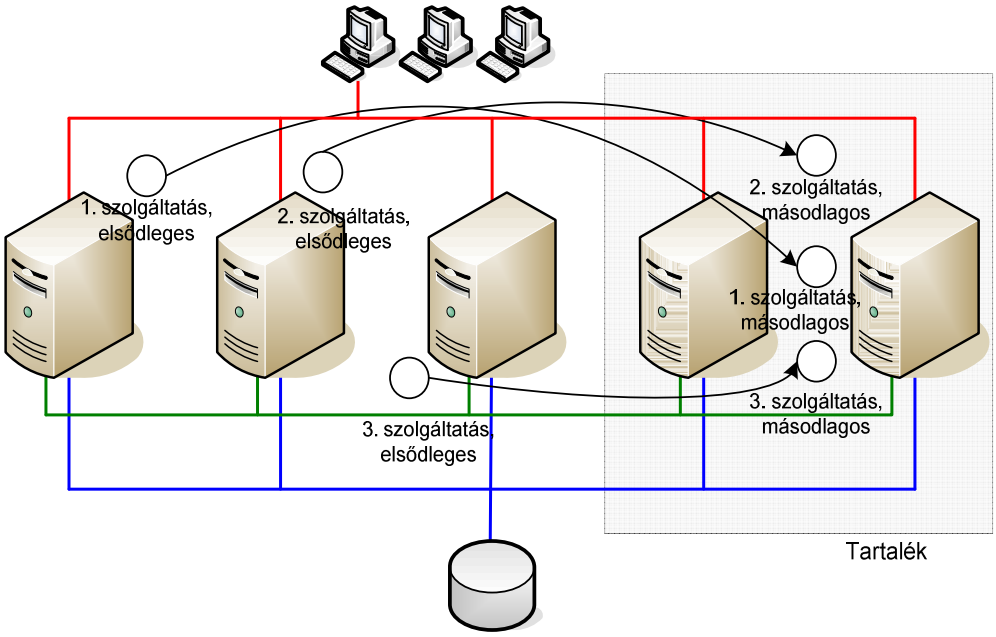
A forró tartalék vagy N+1 (N aktív számítógép, 1 tartalék) topológiában mindegyik szolgáltatás a saját fürttagján fut, meghibásodás esetén a feladatátvétel ugyanarra a tartalék számítógépre történik. Ez a konfiguráció is viszonylag könnyen konfigurálható és kezelhető, de a tartalék számítógépet úgy kell méretezni, hogy akár mindegyik elsődleges fürttag meghibásodása esetén is képes legyen futtatni a szolgáltatásokat. Szükség esetén ebben az esetben is többszörözhetők a szívverések továbbítására és az ügyféloldali hozzáférésre használt hálózatok, valamint a közös adattároló elérési kapcsolatai.



17. ábra: A forró tartalék topológia

3.5.4 N+I topológia

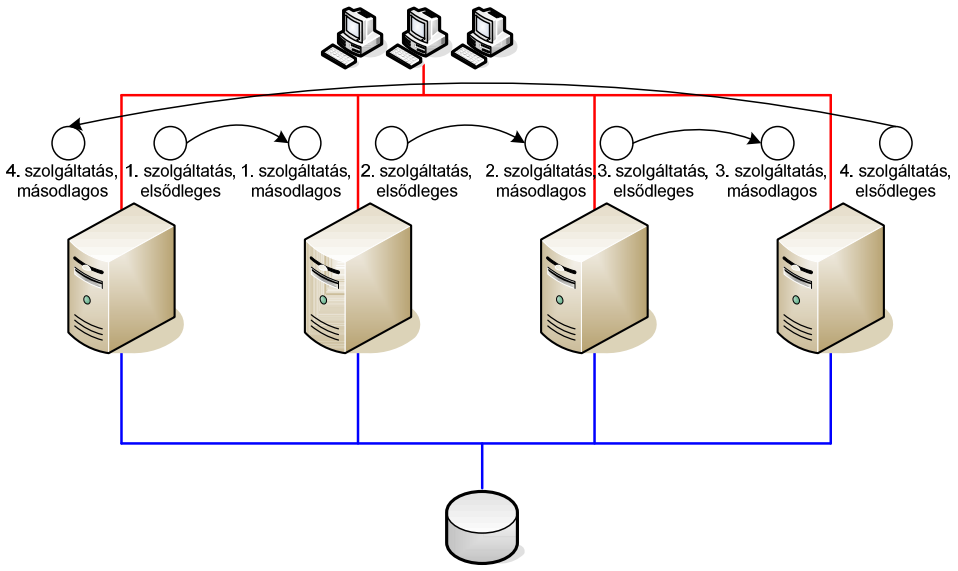
Az N+I topológiában az N számú aktív fürttagra I számú tartalék számítógép jut, hiba esetén a szolgáltatások – a házirendtől függően – a tartalék gépek bármelyikére kerülhetnek. Ennél a megoldásnál egyrészt könnyű a kapacitástervezés, és a többszörös hibák is jól kezelhetők, másrészt ezeket az előnyöket a kihasználatlanul várakozó fürttagok viszonylag nagy számával kell megfizetni.



18. ábra: Az N+I topológia

3.5.5 Feladatátvételi gyűrű

A feladatátvételi gyűrűben mindegyik fürttag az előtte lévőnek a tartaléka, valamint az utolsó fürttagról az elsőre kerülnek a szolgáltatások hiba esetén. Főként több kisebb alkalmazás fürtözésekor használható, és viszonylag könnyű kapacitástervezést tesz lehetővé; ugyanakkor több fürttag meghibásodása esetén egyenetlen terheléseloszlást eredményezhet, és a rendszergazda számára nehéz lehet a topológia áttekintése.



19. ábra: A feladatátvételi gyűrű topológia

3.5.6 Egyéb topológiák

Szükség esetén további topológiák is kidolgozhatók:

- Véletlenszerű feladatátvétel: A rendszergazda nem határozza meg, hogy hiba esetén melyik szolgáltatás melyik fürttagra kerüljön, a véletlenszerű választás statisztikai jóságában bízik. A módszer egyszerű, hiszen nem igényel tervezést, ugyanakkor nem is teszi lehetővé a kapacitástervezést. Főként akkor használható, ha a fürt nagyszámú kisebb szolgáltatást futtat.
- Manuális feladatátvétel: A rendszergazda határozza meg, hogy hiba esetén mi történjen, illetve további, egyedi szabályokat ad meg.

3.6 Jellegzetes problémák a fürtökben

A fürtök kapcsán ki kell emelni néhány jellegzetes, elméleti problémát.

- **Tudathasadás** (split brain): Tudathasadás akkor történik, ha a fürt a hálózati kapcsolat megszakadása miatt két részre bomlik, és mindkét rész azt hiszi, hogy a másik rész meghibásodott. Ilyenkor mindkét rész fogadja az ügyfelek kéréseit és megpróbál hozzáférni a közös adattárolón lévő adatokhoz. Mivel a fürtöt nem ilyen működésre terveztük, a tudathasadás eredménye adatsérülés, adatvesztés lehet.

A tudathasadást mindegyik létező fürtszoftver meg tudja előzni azzal, hogy a megfelelő eszközökkel minden esetben garantálja, hogy a fürt valamelyik része többséget, *quorumot* alkosson; ez a fürtszoftver egyik alapfunkciója. A fürtnek mindig csak a többséget alkotó része viheti tovább a szolgáltatásokat, a másik résznek le kell állnia, és nem szabad használnia az erőforrásokat. Egyes megoldásokban az adatok, erőforrások védelmét kifejezett kizárással is biztosítják.

A többség meglétét sok megoldásnál egy erre a célra kijelölt merevlemezzel, a quorumlemezzel biztosítják. Egy lehetséges eljárás az, hogy ha a fürt két részre bomlik, akkor az a része működik tovább, amely hozzáféréssel bír a quorumlemezhez.

- **Amnézia** (amnesia): Az amnézia kialakulásának menete a következő. Adott egy fürt például két fürttaggal. Az első gép meghibásodik, a második rendben átveszi tőle a feladatokat. A rendszergazda leállítja az első gépet, megjavítja, majd újra üzembe helyezi, ám ekkor a második gép is meghibásodik. Ha időközben bármilyen konfigurációmódosítás történt, akkor az első gép elavult információkkal rendelkezik a fürtről, és nem szabad aktiválni rajta a szolgáltatást.

  Az amnézia kialakulása azzal kerülhető el, hogy a fürtkonfigurációt a közös adattároló eszközre írjuk, és lehetővé tesszük a fürthöz csatlakozó vagy a fürt első tagjaként induló számítógép számára a konfigurációs adatok elérését. Az amnéziát időbeli particionálódásnak (partiton in time) is nevezik [22].

- **Szoftverfrissítés**: Időről időre szükségessé válhat a fürttagokon futó operációs rendszer és alkalmazások frissítése. A művelet két szempontból kritikus, egyrészt azért, mert ha a frissítéssel megváltozik az alkalmazások viselkedése, akkor az újabb és a régebbi változat együttélése problémákat okozhat, másrészt azért, mert a frissítés véglegesítése sokszor újraindítást tesz szükségessé, ami szolgáltatáskieséshez vezet – csakhogy a fürt célja éppen ennek az elkerülése. A fürttagok frissítését gördülő frissítéssel (rolling upgrade) szokás végezni. Ennek során a rendszergazda mindig kiléptet egy-egy fürttagot, elvégzi a frissítését, majd újra üzembe helyezi. Ahogy a művelettel végighalad az összes fürttagon, úgy „végiggördül” a frissítés az összes gépen, innen származik az elnevezés.

- **Egyszeres hibapontok** (single point of failure, SPOF): A fürtépítés során a cél a rendelkezésre állás növelése, tehát olyan struktúrát kell kialakítani, amelyben nincs egyszeres hibapont, vagyis olyan elem, amelynek a meghibásodása a teljes fürt leállásához és a szolgáltatások működésének megszakadásához vezetne. Ennek érdekében nem elég több számítógépet beléptetni a fürtbe, de az összes hálózati kapcsolatot meg kell kettőzni, ideértve az ügyfelek hozzáférését biztosító internetkapcsolatot is, az adattárolás céljára pedig redundáns alrendszert kell választani (az adattárolás általában RAID-5 köteteken történik). Ha az egyszeres hibapontokat tágabb értelemben kezeljük, akkor a környezeti hibaforrásokat is figyelembe kell vennünk, gondoskodva a redundáns áramellátásról, a redundáns légkondicionálásról stb.

3.7 Fürtözés magasabb szinten

Feladatátvételi fürt építésekor az egyszeres hibapontok kiküszöbölésében viszonylag korlátozottak a lehetőségek. Bár maga a fürt redundáns struktúra, az adattárolás csak egy példányban történik. Az adattároló alrendszer lehet ugyan redundáns belül, a lemezmeghajtók szintjén, ám alrendszerből, lemezszekrényből csak egy van, ennek meghibásodása vagy sérülése rendszerleálláshoz vezet. Ugyancsak egyszeres hibapont a környezet, az a helyiség vagy épület, ahol a fürt található.

Ez a felismerés vezetett a fürtözés kiterjesztéséhez és a többszintű fürtözéshez.

3.7.1 Elosztott fürtök

3.7.1.1 Telephelyen belül elosztott fürtök

A környezeti problémák elleni védekezés érdekében a fürtöket úgy is telepíthetjük, hogy két vagy akár több részre osztjuk őket, és a részeket ugyanazon telephely különböző pontján helyezzük el. Ebben az esetben különleges megoldásra nincs szükség, mindössze olyan hálózati és adatelérési eszközöket kell választani, amelyek – például túl nagy késleltetéssel – nem okoznak problémát a fürt működésében. Természetesen a szétosztásnak csak akkor van értelme, ha az adatokat is mindkét helyen tároljuk.

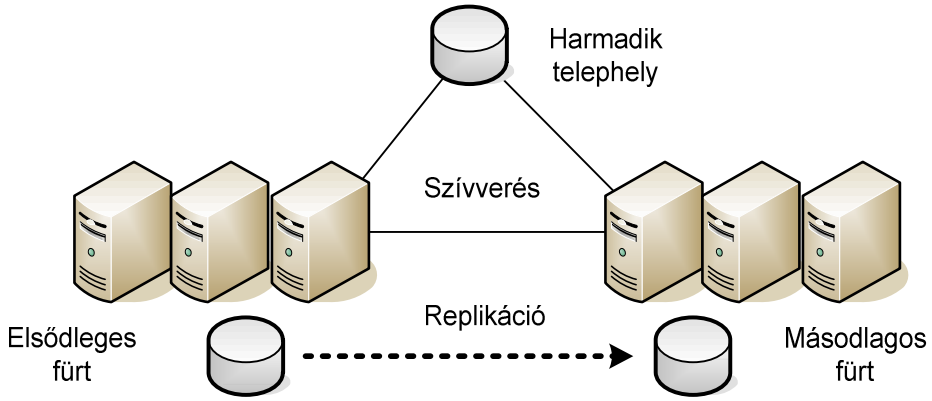
3.7.1.2 Telephelyek között elosztott fürtök

Ha az egyes telephelyeket érintő katasztrófák ellen is védekeznünk kell, akkor telephelyek között kell szétosztanunk a fürt részeit. Ez a megoldás is hasonló az előbbihez, azonban a nagyobb távolságok miatti késleltetések miatt kiemelt figyelmet kell fordítani például a szívverések továbbítására és az adatreplikáció folytonosságára. Előnye ugyanakkor, hogy a távoli telephelyen lévő fürtrész helyi hozzáférést nyújthat az ottani ügyfeleknek.

3.7.2 Többszintű fürtök

Többszintű fürtözésnél a cél szintén az egy-egy telephelyet érintő katasztrófák túlélése. A többszintű fürtökben a szolgáltatást az elsődleges fürt nyújtja, miközben a tartalék fürt várakozik. A fürtök szívverésekkel jelzik egymásnak a működőképességüket. Az elsődleges fürt meghibásodása esetén a másodlagos fürt veszi át a feladatokat. A többszintű fürtök akár kettőnél több telephelyre is kiterjedhetnek.

A többszintű fürtözés érdekessége, hogy az egyszerű fürtözésre jellemző problémák ismét megjelennek; gondoskodni kell a tudathasadás és az amnézia megelőzéséről, valamint lehetőséget kell adni a szoftverfrissítések praktikus elvégzésére.



20. ábra: Három telephelyre kiterjedő fürt

Többszintű fürt felépítésére mutat példát a fenti ábra [11]. A rendszer két fürtből áll, egy elsődlegesből és egy másodlagosból. A harmadik telephelyen kiszolgáló nincs, csak egy quorumlemez, amely akkor jut döntőbírói szerephez, ha valamelyik fürt kiesik. Ugyan a többszintű fürt harmadik helyszínen lévő quorumlemez nélkül is működőképes lenne, a lemez beiktatása további segítséget nyújt a tudathasadás megelőzéséhez arra az esetre, ha megszakadna a hálózati kapcsolat a két fürt között.

3.8 Megvalósítások

Bár a különféle gyártók fürtszoftverei ugyanazokra az alapötletekre épülnek, a megvalósítások közel sem egységesek, mindegyik szereplő különböző ötletek bevetésével igyekszik vonzóbbá tenni a saját termékeit. A későbbiek során Microsoft termékekkel épült tesztrendszerekről lesz szó; azonban szeretném elkerülni, hogy csak egyetlen gyártó megoldását tárgyaljam, ezért az alábbiakban a Sun, az Oracle és a HP egy-egy termékét is áttekintem, valamint az egyik linuxos fürtszoftverről is adok egy rövid leírást.

3.8.1 Sun

A Sun fürtözési megoldása Sun Cluster névvel, jelenleg 3.2-es verziószámmal érhető el, és a cég Solaris operációs rendszere alatt futtatható. A maximális fürtméret architektúrától függ, SPARC processzoros gépekből legfeljebb 16 tagú, x86 processzoros gépekből pedig legfeljebb 4 tagú fürt építhető vele. Terheléselosztásra és feladatátvételre egyaránt alkalmas, bár a Sun dokumentációjában az utóbbi kap nagyobb hangsúlyt [12].

A továbblépéshez meg kell ismerni az adatszolgáltatás fogalmát. A Sun Cluster esetében adatszolgáltatás az a szolgáltatás vagy alkalmazás, amelyet a megfelelő konfigurációval kiegészítve alkalmassá tettünk a fürtözött futtatásra. Két típusa van, a skálázható adatszolgáltatás akár több példányban is futtatható, ilyen szolgáltatásokkal terheléselosztás valósítható meg, míg a feladatátvételi adatszolgáltatások a nagy rendelkezésre állású fürtökön, mindig egy példányban futnak.

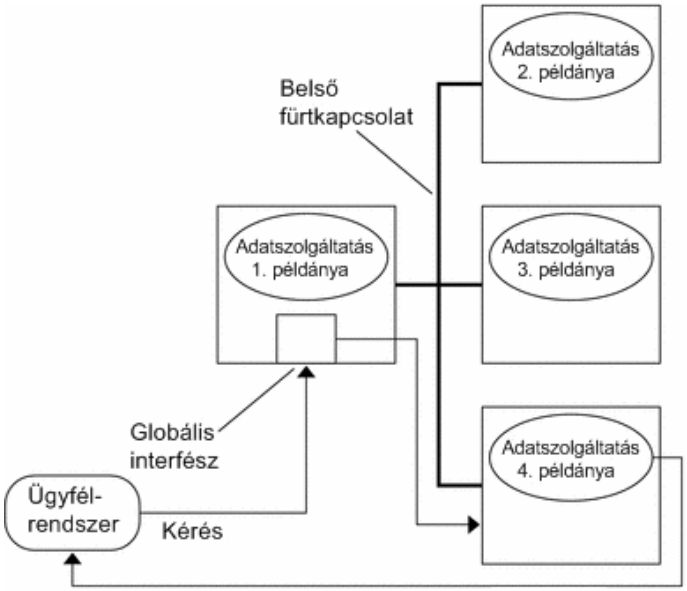
3.8.1.1 Terheléselosztás

A Sun Clusterrel megvalósított terheléselosztási fürtök felépítését a 21. ábra szemlélteti.

Terheléselosztási konfigurációnál az ügyféloldali kérések fogadását végző globális interfészt fenntartó fürttag (proxygép) feladata a kérések továbbítása a többi fürttag felé. A terheléselosztást a terheléselosztási házirend szabályozza.

A skálázható adatszolgáltatások két fő csoportra oszthatók, a „pure” (tiszta) és a „sticky” (tapadós) szolgáltatásokra.

A „pure” szolgáltatásoknál az ügyfél kérésére bármelyik fürttag megadhatja a választ, további megkötés nincs. Ezeknél a szolgáltatásoknál súlyozott terheléselosztás történik, amely alapesetben egyenlően osztja el a kéréseket az egyes fürttagok között.



21. ábra: Terheléselosztás a Sun Clusterrel

A sticky szolgáltatások jellegzetessége, hogy a különböző TCP-kapcsolatok felett létesített alkalmazásszintű kapcsolatok számára lehetővé teszik a kapcsolatadatok megosztását. A sticky szolgáltatások között kétféle típust különböztethetünk meg; az „ordinary sticky” (normál tapadós) szolgáltatások esetében a kérések ugyanazon a porton keresztül futnak be ugyanahhoz a kiszolgálópéldányhoz, míg a wildcard sticky (kb. tetszőleges tapadón) szolgáltatásoknál ez dinamikusan hozzárendelt porton keresztül történik.

3.8.1.2 Feladatátvétel

A feladatátvételi fürtözés terén a Sun Cluster sokoldalúnak mondható, a dokumentációjában külön tárgyalják az egy helyen létesített, a telephelyen belül elosztott és a nagyobb földrajzi távolságra elosztott fürtök létesítetését. A tervezés során itt is központi szerepet kap a szavazati többség (quorum) és a többség kialakításában részt vevő eszközök (quorumeszközök).

Quorumeszköz lehet egy legalább két hoszt által elérhető bármely eszköz. A quorumeszköz szavazattal járul hozzá a szavazati többség kialakításához. Quorumeszközként a következők használhatók:

- több számítógép által elérhető merevlemez, ha támogatja a SCSI-3 protokoll feletti foglalást,
- kéttagú fürt esetében kettős hozzáférésű lemez, ha támogatja a SCSI-2 protokoll feletti foglalást,
- a Network Appliance cég NAS (hálózati adattároló) terméke; a fürt a TCP/IP protokoll feletti foglalásra is képes,
- a SUN Cluster Quorum Server szoftvert futtató számítógép; egy-egy számítógépen több quorumkiszolgáló is futtatható.

A quorum egyszerű szavazati többséget jelent, a többség megszerzése szükséges a működés folytatásához. A szavazati többséget a fürt kiszolgálói és a quorumeszközök együtt alakítják ki. Ebből fakadóan a fürt működőképességének megőrzéséhez nem szükséges a quorumlemez vagy a quorumlemezek elérhetősége, amennyiben a fürtben lévő számítógépek hibátlanok, önmagukban is képezhetnek többséget. Ebből fakad, hogy kettőnél több tagú fürt esetében nem kötelező quorumeszközt telepíteni; kéttagú fürtnél a többség két szavazatot jelent, tehát ebben az esetben a quorumeszköz nélkülözhetetlen. Egy fürtben több quorumeszköz is lehet.

A szavazás során fontos szerep jut a szavazatok mennyiségének. Alapesetben minden fürttag egy-egy szavazatot kap, egy quorumeszköz pedig N-1 szavazatot kap, ha N fürttag kapcsolódik hozzá. A szavazatszámokat a rendszergazda módosíthatja, így nagyobb szabadságot kap a fürt konfigurálásához – ugyanakkor helytelen beállításokkal elérhető, hogy meghibásodás esetén két azonos szavazatszámmal rendelkező részre osztódjon a fürt, amely esetben szavazati többség híján a fürt nem tudja ellátni a feladatát.

A fürt a Cluster Configuration Repositoryban, egy elosztott, a fürttagok által kezelt konfigurációs adatbázisban tárolja a beállításokat. A tagság figyelése szívverésekkel történik, az egyes szolgáltatások működését pedig szondák (probe) figyelik. Szükség esetén a fürt a feladatátvétel lehetősége mellett újra is tudja indítani a szolgáltatásokat. Hasonló módon a megkettőzött hálózati és lemezelérési útvonalak közötti váltásra is van lehetőség, valamint az ügyfélelérés is biztosítható akár több csatolón keresztül is, amelyek aktív-passzív és aktív-aktív konfigurációban is működtethetők.

A fürtszoftver alapvetően erőforrásokkal dolgozik. Az erőforrás egy erőforrástípus értékekkel ellátott és felkonfigurált példánya; az erőforrástípus olyan tulajdonságok összessége, amelyek egy alkalmazást írnak le a fürt számára. Az erőforrások kezelése általános sablonnal vagy a fürttel való szorosabb együttműködést segítő API-kon keresztül történik. Az erőforrások erőforráscsoportokba sorolódnak, az erőforráscsoport a feladatátvétel egysége.

A többes hozzáférésű adattároló eszközöket a fürtszoftver az eszközcsoportokkal kezeli. Az erőforráscsoportok függhetnek az eszközcsoportoktól, például amiatt, hogy egy adott szolgáltatás működéséhez szükség van a fájlrendszer elérésére. Az

erőforráscsoportok és az eszközcsoportok egymástól függetlenül is mozgathatók a fürttagok között, még akkor is, ha előbbi függ az utóbbitól.

Amint a fentiekből is kitűnik, a Sun Cluster nem kifejezetten feladatátvételi vagy terheléselosztási fürt. Egyszerűen erőforráscsoportokkal és eszközcsoportokkal dolgozik, az adatszolgáltatás jellege dönti el, hogy a fürt végül milyen szerepet lát el. Ebből fakadnak az olyan első látásra furcsa szituációk, mint amikor a terheléselosztásra használt, több példányban is futó webkiszolgáló adatszolgáltatás a feladatátvételi erőforrásként kezelt, mindig csak egy példányban létező hálózati címtől függ. Ha szigorúan terheléselosztásban és feladatátvételben gondolkodunk, akkor a korábbi megoldások fényében ez a függés legalábbis szokatlannak hat; ha viszont a Sun Cluster szemléletéhez igazodva mindent erőforrásként kezelünk, akkor tökéletesen illeszkedik a struktúrába.

3.8.1.3 További lehetőségek

A Sun Cluster alkalmas telephelyi fürtök építésére is. Valójában a fürtszoftver részéről semmilyen külön funkció nem segíti ezt – hiszen a korábbiakkal összhangban (Telephelyen belül elosztott fürtök) nincs is szükség ilyenre – azonban a dokumentációban külön tárgyalják a témát, és különféle tervezési mintákat is ismertetnek [13].

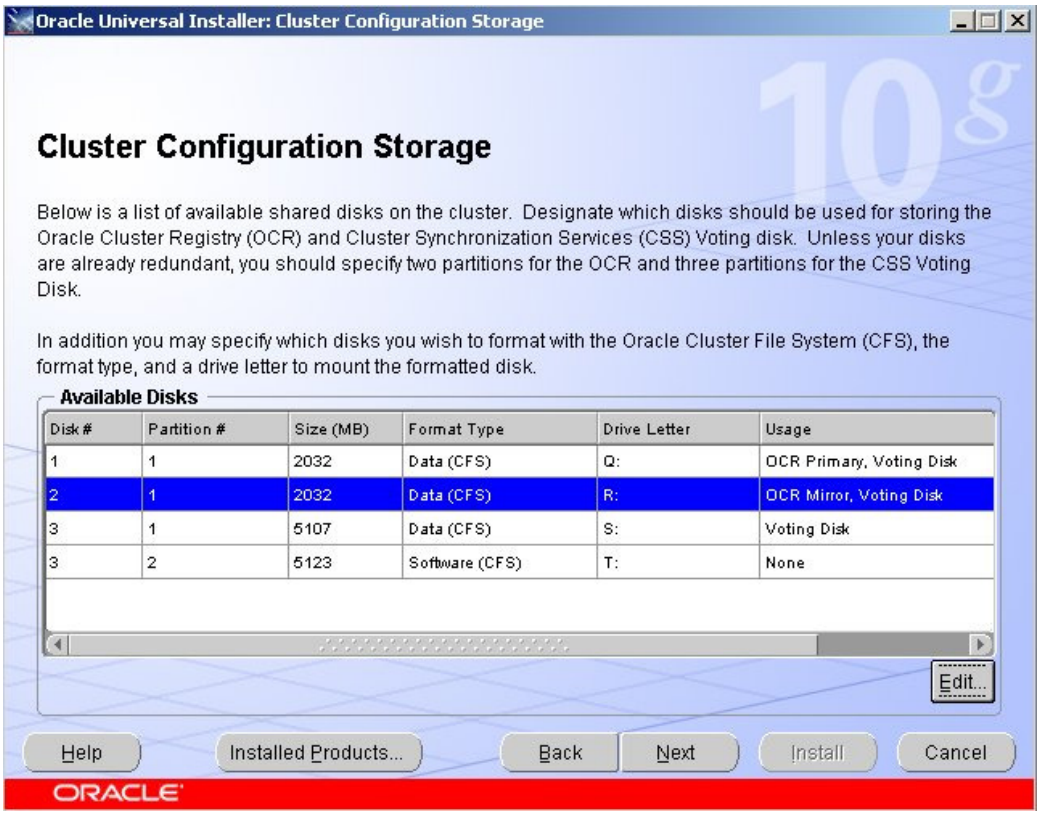
Külön verzió viszont a Sun Cluster szoftvertől függetlenül, a fölé telepíthető Geographic Edition, amely a kiterjedtebb természeti katasztrófák elleni védelmet nyújt azzal, hogy lehetővé teszi a szolgáltatások áttételét egy földrajzilag elkülönített, másik fürtre. A Geographic Edition segítségével partneri kapcsolatok létesíthetők a fürtök között, a partneri kapcsolatok határozzák meg a feladatátvételi lehetőségeket. Egy-egy fürt akár több partneri kapcsolatban is részt vehet. A partneri kapcsolatban álló fürtök szívveréseket továbbítanak a két telephely között, folyamatosan figyelve egymás állapotát. A szívverések továbbítása alapesetben a TCP/IP protokollkészlettel történik, de beépülő modullal további átviteli módok is használhatók, a szívverések akár e-mailben is továbbíthatók. A fürtök között másodlagos szívverés-továbbítási csatorna is létrehozható, például az alapértelmezett csatorna kiesésekor akár telefonmodemen is átvihetők a szívverések. Az egyes fürtök a különböző partneri kapcsolatokban eltérő szerepet is játszhatnak, tehát adott alkalmazáshoz elsődleges, egy másik alkalmazáshoz pedig másodlagos futtatási fürtként is megadhatók.

A partneri kapcsolatban lévő fürtök védelmi csoportokban kezelik a nagy rendelkezésre állásúvá tenni kívánt alkalmazásokat, a védelmi csoportok a szükséges erőforrások mellett például az adatok replikálásával kapcsolatos beállításokat is tartalmazzák. Bár az adatok replikálása nem integráns része a fürtszoftvernek, magát a replikációt a fürt külső szoftverre bízza, felügyeleti szempontból szorosan együttműködik a replikálási megoldással. Telepítésekor a Geographic Edition a replikálás támogatása céljából replikálási erőforráscsoportokat hoz létre a Sun Cluster szoftverben, továbbá az eszközcsoportokat is bővíti a replikálás kezelésével [14].

3.8.2 Oracle Real Application Clusters

Az Oracle termékei között két fürtözési megoldás is található. Az Oracle Clusterware segítségével normál, megosztott elem nélküli feladatátvételi fürt építhető, amely nemcsak az adatbázisok, de az egyéb szolgáltatások, termékek hibavédelmére is alkalmas.

A fürt két alapvető eleme a *szavazólemez* (voting disk) és a *konfigurációs adatbázis* (Oracle Cluster Registry). Az elnevezés ellenére a fürtszoftver mindkettőt fájlként tárolja, és mindkettőnek megosztott lemezen kell lennie. Szavazólemezből több is lehet a fürtben, sőt, az Oracle kifejezetten javasolja több szavazólemez hozzáadását. A szavazólemez akkor jut szerephez, ha például hálózati hiba miatt szavazati többséget kell kialakítani a fürtben, illetve el kell dönteni, hogy melyik tagok vehetnek részt a fürt további működtetésében. A konfigurációs adatbázis a fürt és a fürtözött adatbázisok beállításait tartalmazza, ennek esetében is a több példányban való tárolás az ajánlott. A többpéldányos tárolásnak köszönhetően a meghibásodott, elérhetetlenné vált példányokat a fürt a működés megszakítása nélkül is pótolni tudja.



22. ábra: A szavazólemezek és a konfigurációs adatbázis elhelyezése az Oracle Clusterware telepítésekor

Az Oracle másik fürtözési megoldása a Real Application Clusters (RAC). Érdekessége, hogy – például a tagsági kép fenntartása vagy a szívverések továbbítása céljából – a Clusterware által biztosított infrastruktúrára támaszkodik; a telepítését is a Clusterware beüzemelését követően kell elvégezni. A RAC a megosztott lemezes modellt követi; míg a feladatátvételi fürtben egy-egy adatbázist mindig csak egy adatbázis-kiszolgáló

érhető el, a RAC esetében több adatbázis-kiszolgáló is dolgozhat ugyanazzal a fizikai adatbázissal. Éles környezetben a két megoldás nem zárja ki egymást, ugyanazon a rendszeren belül egyszeres és többes hozzáférésű adatbázisok is lehetnek.

A RAC az egymással párhuzamosan üzemelő adatbázis-kezelő példányoknak köszönhetően egyszerre alkalmas a rendelkezésre állás fokozására és a teljesítmény skálázására; a fürt akár 100 tagig is skálázható. A tagok között nemcsak terheléselosztásra, de hiba esetén feladatátvételre is lehetőség van. A feladatátvétel során az ügyfelek által a meghibásodott fürtaggal létesített munkameneteket a fürt szétosztja a többi tag között; ennek során a függőben lévő tranzakciókat a rendszer ugyan visszagörgeti, a munkamenetek ellenben nem szakadnak meg. A RAC fontos előnye, hogy nem igényli az alkalmazások kódjának módosítását, ugyanakkor esetében is számolni kell azzal, hogy egyrészt az adattároló rendszer teljesítménye korlátozhatja a skálázási lehetőségeket, másrészt az adattárolás továbbra is egypéldányos. Ezeket a problémákat a már megismert módon, replikálással lehet kezelni.

Az adatbázis-kiszolgáló többpéldányos futtatása felveti a gyorsítótárak egységességének problémáját (a terheléselosztásnál már találkoztunk ezzel; lásd: Socket Cloning (SC)). Mivel nem zárható ki, hogy több fürttag is ugyanazokkal az adatokkal dolgozik, gondoskodni kell arról, hogy az egyik fürttag által gyorsítótárazott adatokat a többi fürttag is elérhesse. Ezt a feladatot a Cache Fusion látja el, amely a szükséges memóriablokkokat a fürttagok közötti belsőhálózati kapcsolaton keresztül továbbítva bármely fürttag számára elérhetővé tudja tenni a többi fürttag által módosított, de a lemezre még ki nem írt adatokat.

Az adatbázisok rendelkezésre állása kapcsán érdemes megemlíteni a *Recovery Point Objective* (visszaállítási pont célkitűzés, RPO) fogalmát. Az RPO azt fejezi ki, hogy a vállalat mekkora időtartamra kiterjedő adatvesztést tud elviselni úgy, hogy ne szenvedjen komolyabb kárt. A tőzsdei kereskedésnél az RPO nulla, ellenben más szolgáltatásoknál elképzelhető néhány órás vagy akár néhány napos érték is. Ha valahol papírról folyik adatrögzítés, akkor az egynapos RPO is elfogadható lehet, hiszen az adatok ismételt bevitelével viszonylag korlátozott kár mellett helyreállítható az elveszített információ.

Az adatbázisok rendelkezésre állásának növelésére, az adatok replikálására, az ügyfélkapcsolatok szétosztására, feladatátvételére számtalan további megoldás született, ám ezek tárgyalása túlmutat a dolgozatom keretein. A gyártóktól és a független szakértőktől számos érdekes írás érhető el például az ügyfélkapcsolatok feladatátvételének lehetőségeiről [34], a nagy rendelkezésre állású adatbázisokról (Oracle és Microsoft termékekkel) [35], a különféle fürtszoftverek együttműködéséről [6] vagy éppen a földrajzilag kiterjesztett adatbázisok építéséről [36].

3.8.3 HP NonStop kiszolgálók

A HP NonStop kiszolgálói alacsonyabb szintű fürtözést tesznek lehetővé, amelyben a fürtözés feladata a szolgáltatások elől jobban el van fedve, a hibatűrési funkció inkább hardverközeli [23]. Rövid kitérőként érdemes megismerni ezt a megközelítést is, hiszen érdekes és nagyon komoly tudásanyagot, fejlesztési erőfeszítést tükröző színfoltja a fürtözés világának.

A HP NonStop fürtrendszerek 2-4080 csomópontból (logikai processzorból) állhatnak. Felügyeleti célból a csomópontok szegmensekbe tagozódnak, minden szegmens 2-16 csomópontból áll, míg a teljes fürtrendszer 255 szegmenset tartalmazhat. A rendszer akár több helyre is szétoszthtató, ebben az esetben WAN-kapcsolat biztosítja az egyes részek közötti összeköttetést.

A fürt egyszeres hibapont nélküli, aktív redundáns rendszer, amelyben minden részegység aktívan működik. Az aktív redundancia előnye, hogy késleltetés nélküli feladatátvételt tesz lehetővé, illetve nem fordulhat elő, hogy a tartalék részegység rejtett hibájára csak akkor derül fény, amikor az aktív részegység meghibásodása miatt át kellene vennie a feladatokat.

A szolgáltatások jellegzetes futtatási módja a szolgáltatási párok létrehozása. Ekkor a szolgáltatás két, egymással együttműködő példánya két különböző csomóponton fut, a példányok folyamatosan ellenőrző pontok elhelyezésével rögzítik az állapotadatokat. Ha az egyik példány meghibásodik, akkor az ellenőrző pontok és a csatolt működés révén a fennmaradó példány automatikusan átveszi a kérések kiszolgálást. A művelet az alkalmazások és a felhasználók számára tökéletesen észrevétlen marad, nem kell tehát az egyéb feladatátvételi fürtöknél látott feladatátvételi, újrakonfigurálási idővel számolni, illetve a fürtözéshez és a feladatátvételhez szükséges bonyolultság sem jelenik meg az alkalmazások szintjén.

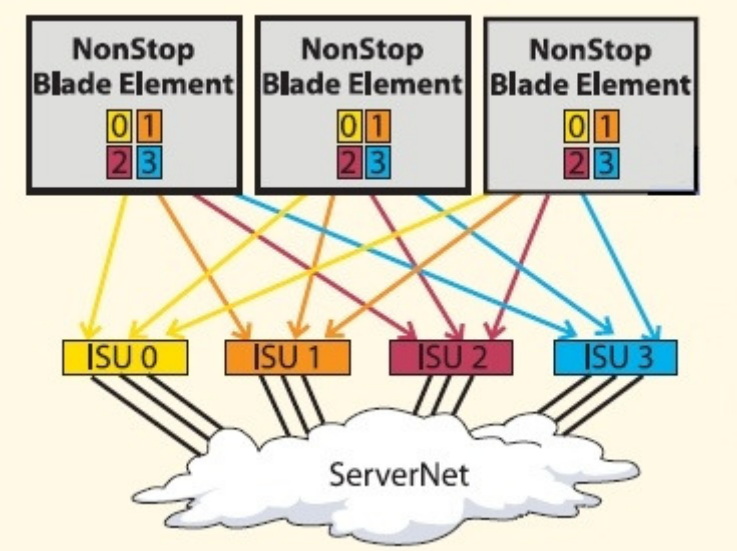
A rendszer széles körben virtualizált abban az értelemben, hogy az alkalmazások és a programozók számára az erőforrások virtualizálva jelennek meg. Így válik lehetővé, hogy a rendszer tökéletesen elfedje az egyes részegységek hibáját. A programozó tehát lemez vagy processzor erőforrással dolgozik; nem kell azzal foglalkoznia, hogy a háttérben milyen RAID-kötet üzemel, abból éppen hány lemez a ténylegesen működőképes, illetve a processzor erőforrás is jelenthet csatoltan működő processzorpárt vagy akár hármas szavazó mögé rejtett processzorhármast is (lásd lejjebb).

A többi fürthöz hasonlóan itt is fontos elem a fürttagok közötti – redundáns – hálózat. Az üzenettovábbítás azonban nem IP protokoll felett, hanem alacsony szintű protokollal folyik, amivel megtakarítható a TCP/IP protokollkészlet miatti többletterhelés. Az ügyfélkapcsolatok hálózatkezelése is redundánssá tehető, lehetőség van arra, hogy adott IP-címet két hálózati csatolón keresztül is kezeljen a kiszolgáló. Ilyenkor a bejövő kéréseket mindig az elsőként beérkező példány alapján fogadja a kiszolgáló, míg a kimenő válaszokat a két adaptert felváltva használva küldi el. Az alkalmazások több fürttagon is futtathatók, ebben az esetben mindegyik tag ugyanazon a címen és portszámon várja a kéréseket, a kéréselosztás – és ezzel a terheléselosztás – round-robin módszerrel történik.

Az adattárolás az egyes fürttagokban RAID-1 köteteken, vagyis tükrözött lemezpárokon történik. Az adatok kezeléséért az adatelérés-kezelő (data access manager) felelős; minden adatkezelési művelet ezen keresztül folyik, valamint az adatelérés-kezelő végzi az általa kezelt adatok szükség szerinti zárolását és gyorsítótárazását is. Az egyszeres hozzáférési pontnak köszönhetően nincs szükség globális zárolásra vagy a gyorsítótárak költséges szinkronizálására [25]. A terhelésnek az adatelérés-kezelők közötti elosztásával a műveletek párhuzamosítása is könnyebb, ami főként az adatbázis-

kezelőknél járhat teljesítménynövekedéssel. Az adatelérés-kezelők párban üzemelnek, két külön fürttagon futó kezelő alkot egy párt. Az egyik példány az elsődleges, a másik a tartalék, a két példány egymással együttműködve rögzíti az állapotadatok ellenőrző pontjait. A virtualizálás elvét követve az adatelérés-kezelők munkája és együttműködése a felhasználó elől rejtve marad.

A NonStop termékcsalád felsőbb kategóriájú tagjainál a processzor erőforrás (a csomópont) kettes vagy hármas szavazóegységekhez (logical synchronization unit, LSU, logikai szinkronizáló modulokhoz) kapcsolódó processzorokat jelent. A processzorok modulokon találhatók, az egyes processzormodulok négy-négy processzort tartalmazhatnak, az egyes szavazóegységekbe bekötött processzorok mindig más modulon helyezkednek el. A szavazóegység kimenete jelenti azt a virtualizált processzort, amelyet a felhasználó lát.



23. ábra: HP NonStop rendszer hármas szavazással. Forrás: [24]

A processzorok csatolása vagy szavazóba kötése lehetővé teszi a számítási eredmények összehasonlítását és az esetleges hibák felismerését. A processzorpárok esetében az eredmény eltérése utal a hibára, a felismerést követően a rendszer megkísérli megtalálni a hiba okát. Ha sikerrel jár, akkor folytatja a működést, illetve letiltja a meghibásodott részegységet, ha pedig nem tudja felderíteni a hibaokot, akkor az adott processzorpárt lekapcsolja. A hármas szavazás ennél is nagyobb redundanciát garantál, esetében még az egyik processzor meghibásodása esetén is fennmarad egy redundáns processzorpár. A fenti ábra alapján is látható, hogy ha a processzor erőforrást egy-egy szavazóegység kimenete jelenti, és a szolgáltatásokat a korábban említett módon párokban futtatjuk, akkor még hardveres meghibásodás és például valamelyik fizikai processzor vagy akár teljes processzor erőforrás kiesése után is redundáns marad a rendszer. Ennek a sokszoros redundanciának köszönhető, hogy a NonStop rendszerekkel – a gyártó szerint – akár hétkilences rendelkezésre állás is elérhető [37].

Szinte magától értetődő, hogy egy ilyen rendszer már nem hétköznapi számítógépekből és nem hétköznapi áron épül fel. Ilyen magas szintű rendelkezésre állást például a pénzügyi szektor igényel és tud megfizetni.

3.8.4 Linux-HA

A nyílt világ, ahogy szinte minden problémára, úgy a fürtözésre is kidolgozta a maga megoldásait. A Linux alá telepíthető fürtök között a kereskedelmi megoldások mellett kereskedelmiből lett nyílt eszközöket, ilyen például az SGI szoftverére épülő Linux FailSafe, illetve a kezdetektől nyíltan fejlesztett rendszereket egyaránt találhatunk. Utóbbira példa a High Availability Linux Project, röviden Linux-HA; az alábbiakban ennek a jellemzőit foglalom össze [30].

A Linux-HA projekt az 1990-es évek végén viszonylag egyszerű célkitűzéssel indult: olyan szívverés-továbbító programot készíteni, amely soros porton keresztül képes követni egy másik számítógép állapotát, és szükség esetén lehetővé teszi a két gép közötti feladatátvételt. Ez a program lett a heartbeat (szívverés), amely fokozatosan további funkciókkal bővült, és amely köré idővel egy teljes fürtkeretrendszer épült. A fejlesztések során a heartbeat kissé túlhízott, ezért egyes funkciókat eltávolítottak belőle; ennek nyomán kialakult a korábban ismertetetthez (lásd: 3.4.2, A fürtök építéséhez szükséges alapszolgáltatások) nagyon hasonló struktúra.

A Linux-HA működése a korábbi megoldásokhoz viszonyítva akár egyszerűnek is nevezhető: a szívverések továbbításával folyamatosan fenntart egy képet a fürttagságról, az egyes fürttagok vagy a rajtuk futó szolgáltatások meghibásodása esetén pedig feladatátvétellel garantálja a szolgáltatások folyamatos elérhetőségét. A Linux-HA közös elérésű adattároló nélkül is működtethető, esetében ilyen adattárolóra kizárólag az alkalmazásadatok tárolása céljából lehet szükség, maga a fürtkeretprogram nem igényel hasonló eszközt. Hiba esetén a heartbeat és a tagsági modul gondoskodik az egységes tagsági kép kialakításáról. A fürtkonfigurációt külön modul kezeli, a heartbeat kommunikációs szolgáltatásaira támaszkodva ez gondoskodik arról, hogy mindegyik tag azonos konfigurációval rendelkezzen. A fürt működésének alapja a többség szavazásos fenntartása.

A Linux-HA néhány érdekessége, jellegzetessége:

- Házirendekkel, függőségekkel kifinomultan szabályozható, hogy mely szolgáltatások mely fürttagokon fussanak, mely szolgáltatások fussanak ugyanazon a fürttagon, illetve mely szolgáltatások nem futhatnak azonos fürttagon.
- A fürttagok maximális száma nincs korlátozva. A dokumentáció szerint 16 tagig biztosan használható, de ennél kétszer nagyobb fürtöt is lehet vele üzemeltetni.
- A heartbeat modul soros kapcsolaton, valamint egyedi címzéses, csoportcímzéses vagy szórásos UDP-csomagokkal tudja továbbítani az információkat. Támogatja az OpenAIS projekt evs kommunikációs protokollját is.
- Az erőforrások kezelése felhasználó által megadott jellemzők alapján is lehetséges, így a feladatátvétel akár tetszőleges szempont alapján is történhet.
- A fürtszoftver külön modullal támogatja a STONITH funkciót.
- Pingeléssel hálózati forgalomirányító vagy kapcsoló működésének figyelése is lehetséges.

3.8.4.1 Az erőforrások elhatárolása

A Linux-HA kapcsán érdemes kitérni az erőforrások és a fürttagok *elhatárolására* (fencing). Az elhatárolás két szinten történhet, a fürttagok elhatárolása az egyes számítógépeket akadályozza meg bármilyen erőforrás elérésében, az erőforrások elhatárolása pedig az egyes erőforrások védelmét biztosítja.

Az elhatárolás elsősorban a közös elérésű adattároló eszközök kapcsán vetül fel. Ha egy fürtben – akár az alkalmazásadatok, akár a fürtállapot tárolása céljából – közös elérésű adattárolóra van szükség, akkor garantálni kell, hogy ehhez az adattároló eszközhöz csak egy fürttag férhet hozzá. Amíg a fürt rendben üzemel, addig ez nem okoz problémát, hiszen a szolgáltatást aktívan futtató fürttag használja a lemezt, a másik fürttag pedig várakozik. Ha azonban meghibásodás történik, akkor a meghibásodott fürttagtól azonnal el kell venni az adattároló használatának jogát; egyrészt azért, hogy a feladatot átvevő fürttag hozzáférhessen a rajta lévő adatokhoz, másrészt azért, hogy a meghibásodott tag ne okozhasson adatsérülést vagy -vesztést.

A feladatátvételi fürtökben ennek a problémának a megoldására általában SCSI-parancsokat használnak. A közös elérésű adattárolót használó fürttag egy foglalási (reserve) parancsot ad ki az eszközre. A lefoglalt eszköz csak a foglalást kiadó számítógép parancsait fogadja el, a többiét visszautasítja. Ha feladatátvétel történik, a hibát észlelő fürttag egy reset paranccsal alapállapotba állítja a megfelelő eszközt vagy a SCSI-buszt, és megpróbálja lefoglalni magának az adattárolót, illetve megvalósítástól függően versengés indulhat a tároló használati jogáért.

A Linux-HA nem használ SCSI-foglalást, ezért esetében különös figyelmet kap az adatok sérülékenysége. Az adattárolók védelmére kétféle megoldás kínálkozik. Az első a meghibásodott fürttag újraindítása a SMONITH szolgáltatással. Ez a módszer kétségkívül hatékony, ám egyrészt kiegészítő hardvert igényel, másrészt, ha a fürttagon több szolgáltatás is fut, akkor egyetlen szolgáltatás meghibásodáskor nem feltétlenül jó megoldás a gép újraindítása. A második megoldás *önmaguk elhatárolására* alkalmas (self-fencing) eszközök használata, ezek önmagukban is képesek arra, hogy csak egy fürttagnak adjanak hozzáférést; ilyeneket például az IBM ServerRAID vezérlői között találunk.

Érdemes megemlíteni, hogy ilyen vezérlővel – éppen az önálló elhatárolás miatt – például Oracle RAC fürt nem üzemeltethető. Elhatárolás szempontjából háromféle eszközt, erőforrást különböztetünk meg:

- külső elhatárolási megoldást és az integritása külső védelmét igényli
- az integritását jellegénél fogva elhatárolás nélkül is megőrzi
- önmaga elhatárolására alkalmas

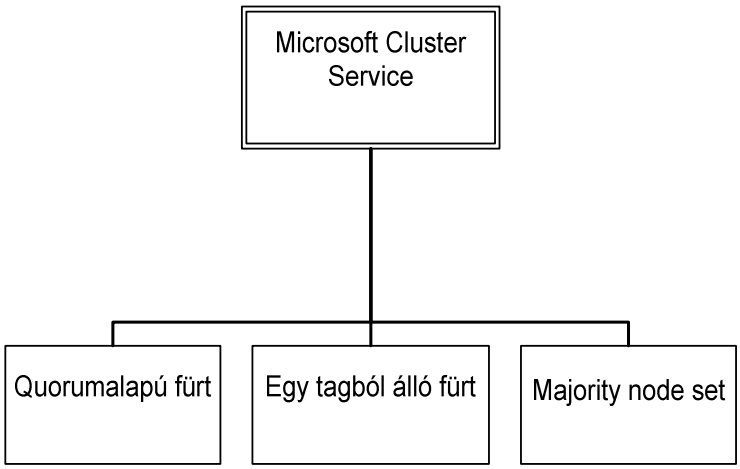
Tervezési kérdés, hogy a fürt milyen erőforrásokat tartalmaz, ám ügyelni kell arra, hogy a különféle típusokat a tervező ne tévessze össze.

3.8.5 Microsoft

A Windows operációs rendszerekben 1997 óta van lehetőség feladatátvételi fürtök építésére. A Windows Server 2003 termékcsaládból az Enterprise és a Datacenter tagokban érhető el ez a szolgáltatás, segítségével legfeljebb 8 tagból álló feladatátvételi fürtök hozhatók létre. A feladatátvételi fürtök a Microsoft terminológiájában *kiszolgálófürtként* (server cluster) jelennek meg; ez az elnevezés házon belül ugyan megfelelő, hiszen a terheléselosztási (load balancing) fürtöktől megfelelő megkülönböztetést biztosít, ám más termékekkel és megoldásokkal összevetve zavaró, ezért a továbbiakban én is tartózkodom a használatától.

3.8.5.1 Fürttípusok

A Microsoft Cluster Service (Microsoft fürtszolgáltatás, MSCS) segítségével háromféle fürt építhető. Az alábbi ábra ezeket foglalja össze [31].



24. ábra: Az MSCS által támogatott fürttípusok

A legegyszerűbb az **egyetlen tagból álló fürt**, amely valójában semmilyen rendelkezésre állási szolgáltatást nem nyújt, kizárólag tesztelésre használható. Fejlesztői szempontból hasznos eszköz, hiszen nem igényli bonyolult, több számítógépből álló infrastruktúra összeállítását.

A **majority node set** (kb. többségi fürttaghalmaz, MNS) fürtök többségi szavazásos alapon működnek. A fürttagok mindegyike saját másolatot tart fenn a fürtkonfigurációról, a fürtszolgáltatás folyamatosan egységesen tartja ezeket a példányokat. A tagok szívverésekkel figyelik egymást, ha valamelyik fürttag meghibásodik, akkor a fennmaradó tagok újraformálják a fürtöt. A fürt működésének előfeltétele a többség fenntartása; a többség itt a fürttagok legalább 51%-át jelenti. Ebből fakad az MNS fürtök hátránya: legalább három tagból kell állniuk, hiszen két tag esetén az egyik tag meghibásodásakor már nem biztosítható a többség. MNS fürtöt – hacsak teljesítményméretezési okok mást nem indokolnak – csak páratlan számú tagból érdemes építeni; például egy három tagból álló fürt ugyanúgy csak egy tag kiesését tudja tolerálni, mint egy négy tagból álló. Az MNS fürttípus alkalmas földrajzilag elosztott fürtök építésére is, esetében csak a fürttagok közötti hálózati kapcsolatot kell biztosítani. Hátránya viszont, hogy az alkalmazásadatoknak a fürttagok közötti

továbbítását (replikálását, tükrözését) nem támogatja, erről külső eszközzel vagy az alkalmazások beépített funkcióival kell gondoskodni.

A Windows Serverben eredetileg található szolgáltatáshoz készült egy kiegészítés, amely fájlmegosztás „tanúként” (witness) való használatát is lehetővé teszi [22]. A tanú szükség esetén a többségi szavazat kialakításához járul hozzá. Szintén újdonság a szívveréskezelés konfigurálhatósága, ami a kevésbé megbízható hálózattal összekötött, földrajzilag elosztott fürtök üzemeltetését könnyíti meg.

A Windows Server 2007 „Longhorn” rendszerben közös elérésű lemezt is lehet majd tanúként használni.

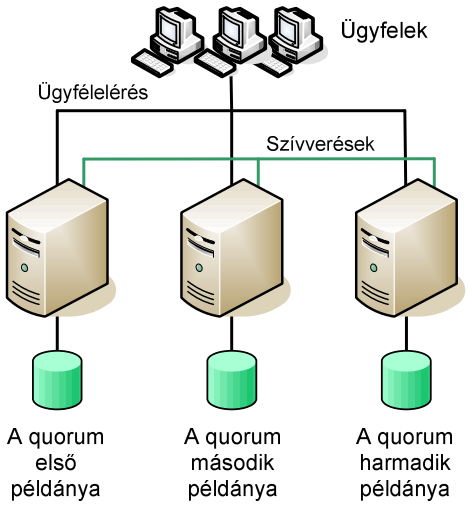
A **quorumeszközre alapuló** fürtökben a fürtkonfiguráció egyetlen példányban tárolódik, a quorumeszközön. A quorumeszköz olyan adattároló eszköz, amelyet fizikailag az összes fürttag el tud érni, de alkalmas arra, hogy mint erőforrást az egyik fürttag ki tudja sajátítani magának, és a többi tag hozzáférését el tudja zárni, így ténylegesen mindig csak egy fürttag érhesse el. Quorumeszköz lehet például kétkapus elérésű SCSI merevlemez vagy tárolóhálózaton (SAN) keresztül elérhető adattároló. A quorumeszközön egyetlen példányban tárolódik a fürtkonfiguráció, az eszköz fizikai lemez erőforrásként jelenik meg a fürtben, amely mindig pontosan egy fürttagon lehet aktív, de a többi erőforráshoz hasonlóan feladatátvétellel vagy átkapcsolással bármelyik fürttagra kerülhet. A quorumeszköz akkor is fontos szerephez jut, ha a fürt azonos számú tagból álló részekre esik szét, ebben az esetben az a partíció folytathatja a működést, amelyik meg tudja szerezni a quorumeszköz használati jogát.

A Longhorn rendszerben lehetőség lesz a fenti két modell elegyítésére, amivel elkerülhető lesz, hogy a quorumeszköz egyszeres hibapontja legyen a fürtnek [38].

3.8.5.2 A quorum

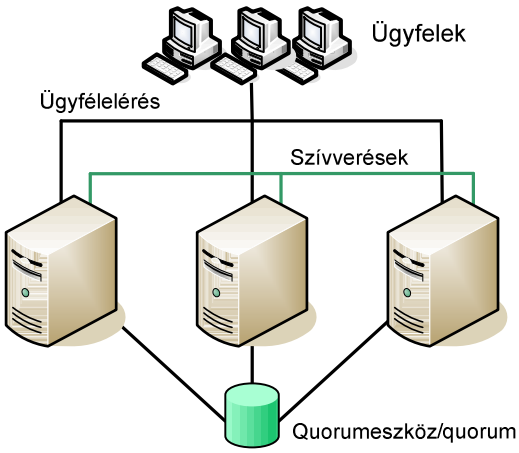
A fentiekből már kitűnik, hogy az MSCS esetében a quorum eltérő jelentéssel bír, mint a többi megvalósításnál. Míg korábban a quorum a szavazati többséget jelentette, az MSCS esetében a quorum a fürtkonfigurációs adatbázist jelenti. A háromféle fürtváltozatban háromféle módon folyik a quorum kezelése.

- Egy tagból álló fürtnél a számítógép helyben tárolja a quorumot.
- MNS fürtnél a quorum fizikailag mindegyik fürttagon megtalálható, de logikai erőforrásként mindig csak az egyik tag kezeli. A quorum minden módosítása csak akkor jut érvényre, ha a tagok legalább 51%-a végrehajtotta.



25. ábra: Az MNS fürt felépítése

- Quorumeszközre alapuló fürtnél a quorum fizikailag egy példányban, a quorumeszközön tárolódik. A quorumeszköz, mint logikai erőforrás mindig csak egy fürttagon lehet aktív, ez a tag végzi a quorum írását.



26. ábra: Quorumeszközre alapuló fürt

Az MSCS annak ellenére, hogy többféle működési módot támogat, mindvégig a megosztott elem nélküli architektúrát követi. Természetesen ez nem zárja ki, hogy például MNS fürt esetében replikációval tartsuk fenn az alkalmazásadatoknak a fürttagok közötti egységességét, ám a fürtkeretrendszer felépítését ez nem érinti.

4 Tesztrendszerek építése

4.1 Tesztkörnyezet

A tesztkörnyezet kialakítására virtuális gépeket használtam, amelyeket a VMware Workstation 5.5.3-as változata alatt futtattam. A gazdagépben 2 GB memória és Intel Pentium 630-as processzor volt. A gazda operációs rendszer Windows XP Professional SP2 volt, vendég operációs rendszerként pedig a Windows Server 2003 rendszer Enterprise kiadását használtam. A virtuális gépek konfigurációja egy SCSI merevlemezt, 2-3 hálózati csatolót, egy processzort, 384 MB memóriát és CD-ROM-meghajtót tartalmazott. A virtuális gépekből az alapesetben meglévő hajlékonylemezes meghajtót, a hangkártyát és az USB-vezérlőt eltávolítottam.

A hálózatelérésre a VMware „bridged" módját használtam, ennél a virtuális gépek hálózati adaptere egy virtuális hálózati kapcsolóhoz csatlakozik, a virtuális gépek úgy jelennek meg a hálózaton, mintha annak az önálló tagjai lennének; minden hálózati adapter saját IP-címet kap.

Mivel fizikai infrastruktúra nem állt a rendelkezésemre, csak virtuális gépekkel dolgozhattam, a mérési eredmények csak iránymutató értékként kezelhetők. A mérések során nem figyeltem, hogy a fürtök újrakonfigurálása mennyi ideig tart, kizárólag az érdekelt, hogy mennyi a szolgáltatások kiesésének időtartama. Összetett infrastruktúrában mért éles értékek – ideértve a fürt újrakonfigurálásának időigényét is – például a Microsoft TechNet egyik cikkében találhatók [3].

4.2 Windows-alapú terheléselosztó fürt

4.2.1 Technikai részletek az előzetes tervezéshez

A Microsoft az interneten és a termékdokumentációban részletes leírást ad a Windows kiszolgálók mindegyikében megtalálható terheléselosztó (network load balancing, a továbbiakban NLB) szolgáltatásról. Éppen ezért nem célom, hogy ezt a dokumentációt másoljam, inkább az alábbiakban csak az érdekesebb technikai részleteket próbálom kiemelni, illetve a megvalósítási tapasztalataimat foglalom össze.

A Windows-alapú NLB fürtök tervezéséhez tisztában kell lennie a szolgáltatás hálózatkezelésével. A hálózatkezelés megértéséhez viszont meg kell érteni az alapszintű hálózati eszközök működését.

4.2.1.1 Keretkezelés a hálózati eszközökben

Az Ethernet hálózatokban minden keretnek van egy forrás és egy cél MAC-címe, ennek alapján történik a továbbítás a hálózati szegmensen belül. A hálózatok összekötésére használt eszközök eltérően kezelik a MAC-címeket.

- A hub nem vizsgálja a MAC-címet. Ha kap egy keretet, akkor az összes portján kiküldi.

- A kapcsoló (switch) megvizsgálja a kapott keret MAC-címét. Ha még nem ismeri a címet, akkor az összes portján kiküldi. Feltételezhető, hogy a keretet fogadó eszköz valamilyen választ is küld, ekkor a válaszkeretben forrás MAC-címként a saját címét tünteti fel. A kapcsoló ennek alapján tudja, megtanulja, hogy az adott MAC-cím melyik portján érhető el. A továbbiakban az erre a címre küldött kereteket kizárólag a megfelelő portján küldi ki. Ha a kapcsoló soha nem kap keretet ilyen forrás MAC-címmel, akkor a címet soha nem tudja megtanulni és porthoz rendelni.
- A forgalomirányító (router) a hálózati rétegbeli cím (IP-cím) alapján továbbítja a csomagokat. A kereteket újra előállítja; a neki küldött keretek célcíme és az általa küldöttek forráscíme a saját MAC-címe.

4.2.1.2 Az NLB szolgáltatás hálózatkezelése

Az NLB fürthöz a létrehozásakor hozzá kell rendelni legalább egy virtuális IP-címet. Az NLB szolgáltatás az erre a címre küldött kéréseket szűri, osztja el, ebből fakadóan ahhoz, hogy egy szolgáltatásra kiterjedjen a fürtözés hatálya, a szolgáltatásnak fogadnia kell az ezen az IP-címen érkező kéréseket. A szűrés minden olyan kérésre kiterjed, amelynek forráscíme eltér a dedikált IP-címtől. A fürt virtuális IP-címére küldött kérésekre az összes fürttag válaszol, illetve válaszolhat.

A fürt minden tagja rendelkezhet egy saját, kizárólagosan használt, dedikált IP-címmel. Ezzel biztosítható, hogy a fürt forgalma mellett a fürttagok egyedi forgalmat is bonyolíthassanak, tehát felügyeleti célból egyenként is megcímezhetők, valamint nem fürtözött szolgáltatások is futtathatók rajtuk.

A fürt tagjai által indított, a fürttől független kérések forrás IP-címe mindig a dedikált IP-cím, és nem a fürt virtuális IP-címe. Erre azért van szükség, hogy a már létrejött párbeszédek során a másik féltől eredő további üzenetekre a terheléselosztás már ne terjedjen ki, ellenkező esetben előfordulhatna, hogy az üzenetek másik fürttaghoz kerülnek, mint amelyikkel a párbeszéd folyik.

A fürtnek küldött kérésekre adott válaszok forrás címe a fürt virtuális IP-címe.

Az NLB fürt talán legérdekesebb részlete a MAC-címek kezelése. A fürt kétféle MAC-címkezelési módot támogat.

Unicast mód:

Az NLB fürtök alapértelmezett módja az unicast (egyedi címzéses) mód. Unicast módban az NLB szolgáltatás minden fürttagon lecseréli a hálózati kártya eredeti MAC-címét. A MAC-címet a szolgáltatás a fürt virtuális IP címéből származtatja, és minden fürttagon azonos az új cím. A bejövő kereteket a MAC-cím egyezése miatt minden csomópont fogadja, majd a felettes rétegben az NLB szolgáltatás megszűri őket.

A kimenő keretek forrás MAC-címe a választ adó fürttaghoz generált egyedi, virtuális cím, amit a fürt közös MAC-címéből a fürttagok a prioritás (lásd lejjebb) felhasználásával generálnak. Mivel a kimenő válaszkeretek forrás MAC-címét az NLB szolgáltatás módosítja, így a hálózati kapcsoló nem tudja megtanulni a fürt MAC-címét, és minden bejövő kérést továbbra is eljuttat minden csomóponthoz. Ez a *switch floodingnak* (a kapcsoló elárasztásának) nevezett „trükk” a szolgáltatás működésének

egyik alapeleme. Az elárasztás miatt a fürtöt érdemes külön kapcsolóra csatlakoztatni, mert ha a kapcsolóhoz több fürt tagjai vagy más, nem fürtözött rendszerek is kapcsolódnak, akkor a folyamatos elárasztás hálózati többletterhelést okoz náluk.

Az unicast módnál a fürttagok közötti kommunikáció megszűnik, mert az elküldendő keretekben a cél MAC-címe azonos lenne a küldőével, ami miatt a csomagok a protokollkészleten belül visszahurkolódnának, és nem kerülnének ki a fizikai hálózatra. Az egyes fürttagok és a külső állomások közötti kommunikáció azonban fennmarad, mert a keretek ugyan minden fürttaghoz eljutnak, de a szétválasztásuk a fürttagok egyedi IP-címe alapján megoldható. A fürttagok közötti kommunikáció második hálózati csatoló telepítésével biztosítható.

Tervezési szempontból lényeges, hogy a kimenő keretek forráscímének hamisítása letiltható. Erre akkor lehet szükség, ha a fürt tagjai hubhoz csatlakoznak, a hálózati struktúra felsőbb szintjén lévő kapcsoló portjait viszont nem akarjuk terhelni a fürtnek szánt forgalommal. Ilyenkor a kapcsolónak a fürt forgalmát csak az egyik portján kell kiküldenie, a szétosztásról a hub, mint alacsonyabb szintű eszköz eleve gondoskodik.

Multicast mód:

Multicast (csoportcímzéses) módban az NLB szolgáltatás egy második, multicast MAC-címet rendel a hálózati csatolóhoz, az eredeti MAC-cím megőrzése mellett. A fürttagok közötti és a külső gépekkel végzett kommunikáció ebben az esetben zavartalan marad, ugyanakkor problémát jelenthet, hogy a fürt virtuális IP-címe egyedi címzéses, és ehhez a fürtszolgáltatás multicast MAC-címet rendel; ezt egyes hálózati eszközök nem támogatják.

A kétféle mód között további eltérés, hogy unicast módban a fürttagok szórással küldik ki a szívveréseket, multicast módban viszont a fürt MAC-címére címzik őket; ezáltal tehermentesülhetnek a hálózat egyéb gépei.

4.2.1.3 Prioritás, portszabályok és affinitás

A fürt minden tagjához tartozik egy prioritás (hosztprioritás). A legkisebb számú fürttag a legnagyobb prioritású, minden olyan ügyfélforgalmat, amelyre nem akarjuk kiterjeszteni a terheléselosztás hatályát, ez a fürttag kezel. Az ilyen kérések tehát mindig ugyanarra a kiszolgálóra futnak be.

A portszabályok lehetővé teszik, hogy bizonyos porttartományokhoz fürttagokat rendeljünk, illetve bizonyos portok forgalmának a fogadását letiltsuk.

Kétféle portszabály adható meg:

- Egyhosztos szabály: Az adott port forgalma a legmagasabb prioritású fürttagra kerül.
- Többhosztos szabály: A bejövő forgalom elosztása az összes kiszolgáló között történik. A kiszolgálókhoz százalékos értékek adhatók, így a nagyobb kapacitású gépekhez több kérést lehet eljuttatni

Az ügyfélaffinitás az ügyfelek és a fürttagok egymáshoz rendelését segíti elő. Ügyfélaffinitás megadására többhosztos szabálynál van lehetőség. Háromféle affinitást különböztetünk meg:

- Nincs affinitás: Az azonos forrás IP-cím különböző portjainak forgalma különböző kiszolgálókra kerül. Ez a legjobb válaszidejű, legjobban elosztott megoldás, hiszen a különféle kérések – például egy weboldal különféle elemeinek a letöltése – optimális esetben párhuzamosan is feldolgozhatók.
- Egy ügyfeles affinitás: Adott IP-cím minden forgalmát a fürt valamely tagjára juttatja.
- C osztályú affinitás: Egy C címosztály minden IP-címének minden forgalmát adott fürttagra juttatja.

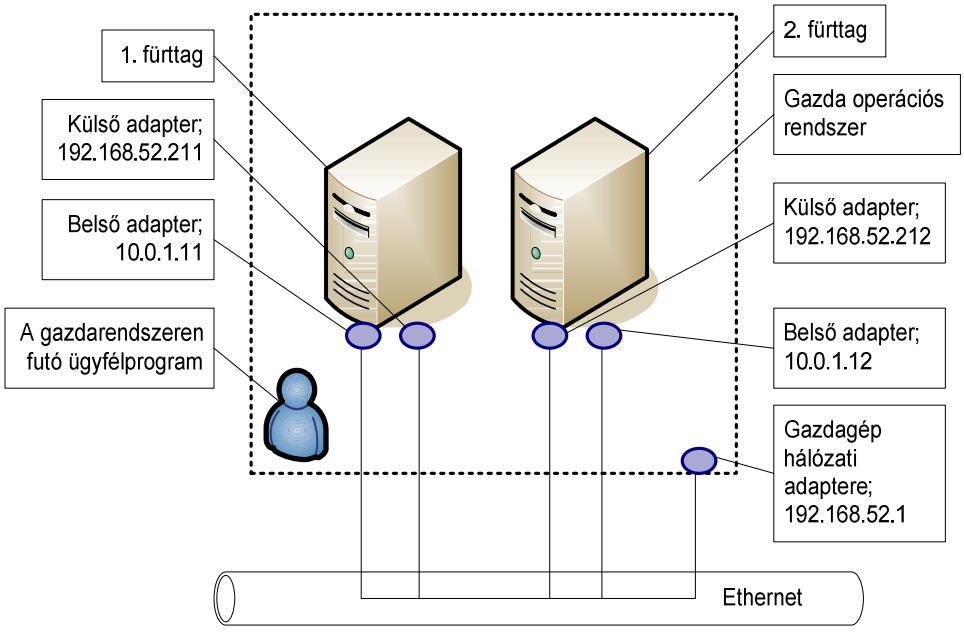
Affinitás használatára akkor lehet szükség, ha az ügyfél állapotát kezelő és követő alkalmazást futtatunk, például bevásárlókosár tartalmát kell kezelnünk. Ilyenkor az affinitással lehet biztosítani, hogy a munkamenet teljes forgalma ugyanahhoz a fürttaghoz kerüljön. Érdemes azonban figyelembe venni, hogy ha a munkamenet ideje alatt kiesik egy fürttag, illetve bővítjük a fürtöt, akkor a fürt újraelosztja a terhelést, és ennek hatására az állapotalapú kapcsolat más fürttaghoz kerülhet, ami a gyakorlatban a kapcsolat megszakadását jelenti.

A C osztályú affinitásra akkor lehet szükség, ha az ügyfél nagyobb méretű hálózatba tartozik, és a kimenő forgalmát több proxykiszolgálón is bonyolíthatja. Ilyenkor előfordulhat, hogy egy-egy munkamenet különböző kérései más proxykiszolgálón keresztül érkeznek – ám ha ezek azonos C osztályú címtartományba tartoznak, akkor megoldható a kérések azonos fürttaghoz való továbbítása.

Ha az állapotadatokat (például cookie használatával) be tudjuk építeni a kérésekbe, akkor nincs szükség affinitásra.

4.2.2 Terheléselosztási tesztkörnyezet

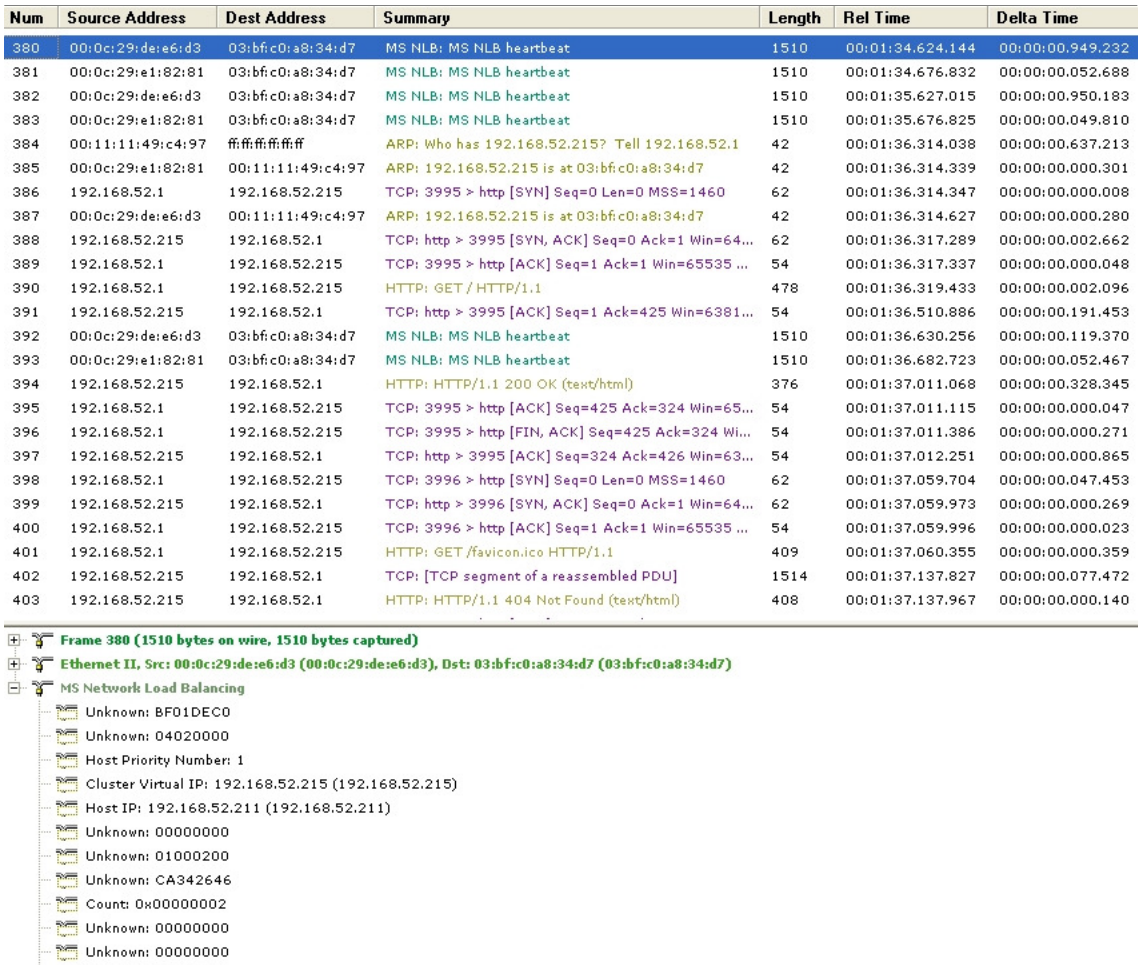
A terheléselosztás kipróbálásához az alábbi struktúrát alakítottam ki.



27. ábra: Terheléselosztási tesztkörnyezet

Mindkét fürttagra telepítettem az Internet Information Services webkiszolgálót, és mindkét tagon elhelyeztem a webkiszolgáló gyökérkönyvtárába egy egyszerű HTML fájlt, amely alapján meg tudtam állapítani, hogy a kérés melyik fürttaghoz jutott. A tesztelés megkönnyítése érdekében a webkiszolgálókon letiltottam a HTTP keep-alive funkciót, az inaktív kapcsolatok fenntartási idejét (Connection timeout beállítás) pedig 1 másodpercre korlátoztam. A fürt virtuális IP-címe 192.168.52.215 lett, az affinitást kikapcsoltam. A hálózati forgalmat a Packetyzer protokollelemző alkalmazással figyeltem [39]. (Lásd: 28. ábra: Képernyőkép a Packetyzer programról. Az alsó ablaktáblán egy szívverési csomag részletei láthatók.)

A fenti struktúrával csak alapszintű tesztelésre nyílt módom, hiszen a kiszolgálóknak és az ügyfeleknek a 2.3.2 pontban említett 1:5 arányát nem állt módomban biztosítani.



28. ábra: Képernyőkép a Packetyzer programról.

4.2.3 Teszteredmények

4.2.3.1 Egy hálózati adapter, unicast mód:

A legegyszerűbb működési mód kipróbálásához a fürttagok belső adapterét letiltottam. A dokumentáció felhívja a figyelmet arra, hogy mivel ilyenkor a fürttagok közötti kommunikációs megszűnik, a fürtfelügyeleti program sem használható egyik fürttagon

sem. Éppen ezért a felügyeleti programot átmásoltam a gazdagépre, és azon is megpróbáltam futtatni. A tapasztalataim a következők voltak:

- Az első fürttag hozzáadásakor nem észleltem hibát. A második fürttag hozzáadását követően viszont valóban megszakadt a kapcsolat a második taggal, és a felügyeleti program nem tudta megállapítani a második tag állapotát.
- A felügyeleti program újra-, illetve elindítva mindkét tagon egyetlen tagból álló fürt meglétét jelezte. A gazdagépen futtatva úgy látszott, mintha a fürt egyetlen tagból, az elsőként hozzáadott tagból állna.
- A saját IP-címén mindkét fürttag elérhető maradt.
- A virtuális IP-címen futó webkiszolgáló csak akkor válaszolt, ha a fürtszolgáltatás az elsőként hozzáadott taghoz irányította a kérést, egyébként a böngésző időtúllépést jelzett.

Összefoglalva: ebben az üzemmódban nem működött helyesen a fürt. A VMware támogatási oldalain tárgyalják ugyan a jelenséget [32], ám a cikk a kiszolgálói célú ESX Server termékre vonatkozik, a benne szereplő megoldás a Workstation terméknél nem alkalmazható.

4.2.3.2 Egy hálózati adapter, multicast mód:

A fürtszoftver ezúttal multicast MAC-címet rendelt a fürthöz. (A fürt címe 03:bf:c0:a8:34:d7 lett, amely [33] szerint valóban multicast cím.) A fürt működésében semmi rendelleneset nem tapasztaltam:

- Mindkét fürttag elérhető volt a dedikált IP-címén.
- A virtuális IP-címre intézett kérdések egy részét az egyik, egy részét a másik fürttag válaszolta meg.
- A felügyeleti programmal a gazdagépről és a fürttagokról is hiba nélkül követni lehetett a fürt állapotát.

Az IGMP-támogatás tesztelésére megfelelő hálózati eszköz hiányában nem volt lehetőségem.

4.2.3.3 Két hálózati adapter, unicast mód

A két hálózati adapteres működés kipróbálásához a fürttagok mindkét hálózati adapterét engedélyeztem. A terheléselosztást a külső hálózati adapteren kapcsoltam be. Tapasztalataim a következők voltak:

- A fürtfelügyeleti program bizonytalanul működött, egyes esetekben nem tudta elérni a fürttagokat.
- A fürthöz intézett webes kérések közül csak azok teljesültek, amelyek az elsőként hozzáadott fürttaghoz kerültek, a második fürttag nem válaszolt.
- A dedikált IP-címén mindkét fürttag megfelelően kiszolgálta a kéréseket.

Az unicast módnál jelentkező hálózati problémákra a termékdokumentáció szerint a második hálózati adapter telepítése lenne a megoldás. Ennek ellenére a várt működést nem sikerült reprodukálni. Lehetséges, hogy ebben a VMware hálózatkezelése is

közrejátszott. A felügyeleti program megbízhatatlan működésében a feltételezésem szerint zavaró tényező lehetett, hogy a programok a névfeloldás során elsőként talált címen próbáltak csatlakozni, és ha ez a 192.168.52-es hálózatra esett, akkor a művelet sikertelenül zárult, függetlenül attól, hogy a hosts fájlban a 10.0.1-es hálózathoz is megadtam a névfeloldási bejegyzést.

4.2.3.4 Két hálózati adapter, multicast mód

Ennél a módnál a második hálózati adapter telepítése elhagyható, megléte csupán a felügyeleti forgalom elkülönítését segíti. Ebben az esetben a várakozásaimnak megfelelően:

- A fürtfelügyeleti program a fürttagokon és a gazdagépen egyaránt megfelelően működött.
- A kérések kiszolgálásában mindkét fürttag részt vett.
- Mindkét fürttag elérhető maradt a dedikált IP-címén is.

4.2.3.5 Mérések

A fürttel szembeni egyik elvárás, hogy az egyik fürttag meghibásodásakor konfigurálja újra magát, illetve tartsa fenn a szolgáltatás folyamatos elérhetőségét. A hibát úgy szimuláltam, hogy a VMware-ben megszakítottam az egyik fürttag külső hálózati kapcsolatát. Eközben a gazdagépről folyamatosan pingeltem a fürt virtuális IP-címét a következő paranccsal:

```
ping -s 1 -w 100 -t 192.168.52.215
```

A -s kapcsoló az időbélyegek kiírását engedélyezi, a -w 100 kapcsoló hatására a ping program 100 ezredmásodpercet vár a válaszra, a -t hatására pedig addig folytatódik a pingcsomagok küldése, amíg a Ctrl+C billentyűkombinációval meg nem szakítjuk.

A szolgáltatás, vagyis a fürt virtuális IP-címének elérhetőségében sem a fürttag kapcsolatának megszakításakor, sem a kapcsolat helyreállításakor nem tapasztaltam fennakadást. Hasonlóan nem volt észlelhető kimaradás a konfiguráció módosításakor – második virtuális IP-cím hozzáadásakor és portszabály módosításakor – sem, valamint fürttag eltávolítása vagy hozzáadása sem okozott kiesést.

4.3 Windows-alapú MNS fürt

4.3.1 Technikai részletek az előzetes tervezéshez

Az MNS fürtök előzetes tervezése során három dologra kell ügyelni.

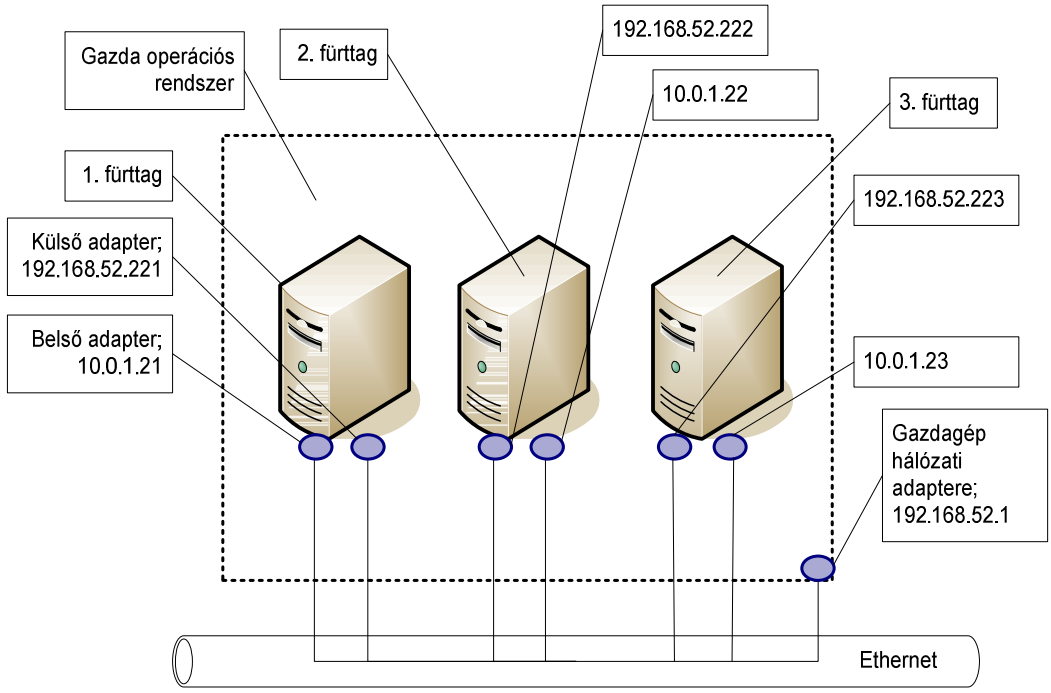
- A többségi szavazat biztosítása. Az MNS fürtök többségi szavazásos alapon működnek. Mivel a működőképesség megőrzéséhez legalább 51%-os szavazati arány szükséges, legalább három tagból kell állnia a fürtnek; ez egyben költségnövelő tényező.
- Az alkalmazásadatok elérhetősége. Sok alkalmazás erőforrás-függőségi fája csak egy fizikai lemez erőforrásra alapozva építhető fel, ezekhez az alkalmazásokhoz

tehát szükség van egy közös elérésű adattároló eszközre. Az MNS fürtöknél azonban – alapesetben – nincs ilyen. Az egyik lehetséges eljárás, hogy bővítjük egy ilyen adattárolóval a fürtöt, a másik, hogy replikációt alkalmazunk. Minden esetben a kiválasztott alkalmazás jellemzőitől függ, hogy milyen megoldás alkalmazható.

- A földrajzi elosztás lehetősége. Az MNS fürtök erőssége, hogy alkalmazásukkal könnyen építhetünk földrajzilag elosztott fürtöt. A fürt két része közötti WAN-kapcsolat esetleges lassúsága, késleltetése, csomagvesztése azonban nem várt hibákhoz, indokolatlannak tűnő feladatátvételekhez vezethet, továbbá a fenntartásának a költségét is figyelembe kell venni.

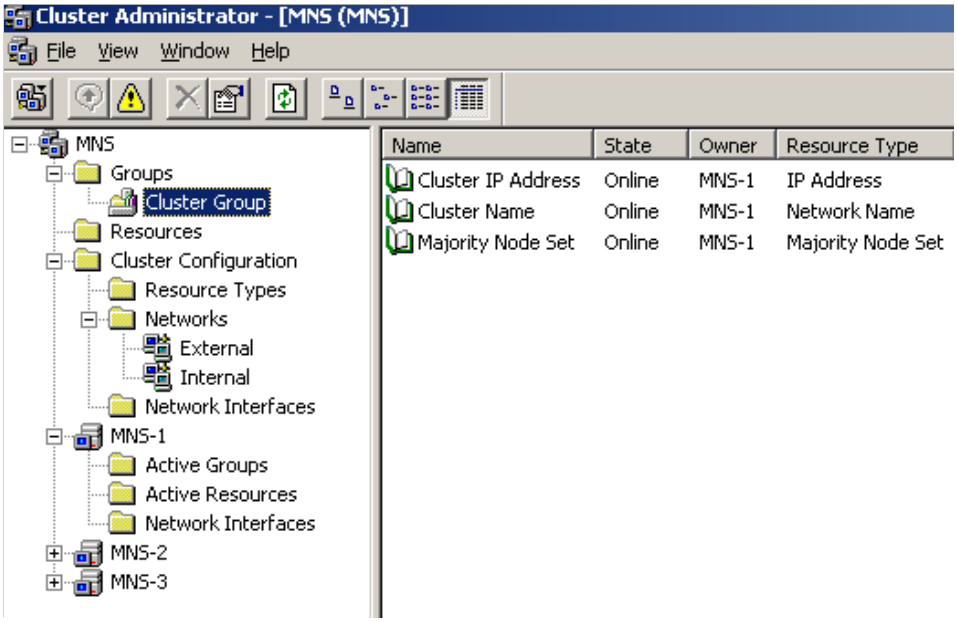
4.3.2 MNS fürtözési tesztkörnyezet

Az MNS fürt kipróbálásához az alábbi infrastruktúrát állítottam össze.



29. ábra: Tesztkörnyezet az MNS fürthöz

A fürtöt úgy konfiguráltam, hogy a 192.168.52-es hálózatot kizárólag ügyfélhozzáférésre, a 10.0.1-es hálózatot pedig kizárólag a szívverések továbbítására használja. A fürt üzembe helyezése után a felügyeleti programban az alábbi ábrán látható erőforrások jelentek meg.



30. ábra: MNS fürt alapelemei a felügyeleti programban

A Cluster IP Address a fürt virtuális IP-címe, ebben az esetben 192.168.52.225 lett. A Cluster Name erőforrás a fürt hálózati neve (MNS), a Majority Node Set pedig a többségi szavazatot megtestesítő erőforrást jelenti. Korábban már utaltam rá (lásd: 3.2.2, Megosztott elem nélküli modell), hogy ennek az erőforrásnak is mindig csak egy gazdája lehet, a fenti ábrán ez az MNS-1 fürttag.

Tesztelési célból *test* névvel létrehoztam egy újabb erőforráscsoportot. Ebbe egy általános alkalmazási erőforrást illesztettem be, *testapp* névvel. A futtatott alkalmazás a Paint rajzolóprogram lett, amelynek fürtözött üzemeltetésére valós környezetben aligha találnánk példát, próbálgatás céljára azonban megfelelt.

4.3.3 Mérések

Első lépésként azt vizsgáltam, hogy a test erőforráscsoport felügyeleti intézkedés hatására mennyi idő alatt kerül át másik fürttagra. Ebben az esetben az erőforráscsoport áthelyezése mindössze az mspaint.exe program leállítását és az új fürttagon való elindítását jelent, illetve a változás regisztrálását. Kísérleteim szerint a művelet kevesebb, mint 1 másodperc alatt lezajlott.

Második próbálkozásként a kezelőprogramból leállítottam a fürtszolgáltatást azon a fürttagon, amelyen a Paint futott, és figyeltem, hogy mennyi idő alatt jelenik meg az alkalmazás valamelyik másik fürttagon. Az eltelt idő 2,5-3,5 másodperc volt.

A Cluster Group erőforráscsoport áthelyezése ennél összetettebb feladat. Egyrészt itt három erőforrással kell számolni, másrészt a fürt neve függ a fürt IP-címétől. A méréseim szerint a csoportnak a felügyeleti programból kezdeményezett áthelyezéseinél 6 másodpercet igényelt a fürt IP-címének online állapotba hozatala, míg a fürt nevének online állapotba kerüléséhez további 6 másodperc kellett. A Majority Node Set áthelyezése a parancsra való kattintáskor nem azonnal, viszont érzékelhető kimaradás nélkül végbement. A tapasztalataim szerint, ha a fürt nevének DNS-regisztrációját is kötelezővé tettem, akkor az átmozgatás teljes időigénye 15 másodpercre nőtt; itt

azonban külső tényezőként megjelent a DNS-kiszolgálótól való függés, ezért a többi mérésnél mellőztem ezt a beállítást.

Utolsó mérésként hálózati hibát imitáltam azzal, hogy a VMware segítségével megszakítottam az egyik fürttag hálózati kapcsolatait. Előzetesen mindkét erőforráscsoportot erre a fürttagra helyeztem. A mérések szerint a hálózati kapcsolat megszakításától 40 másodperc telt el, mire valamelyik fürttagon megnyílt a Paint, a Cluster Group erőforráscsoport összes eleme pedig további 6 másodperc múlva állt online állapotba.

Az eredmények alapján látható, hogy 40 másodperc volt, amíg a fürt észlelte a tag kiesését és elvégezte az újrakonfigurálás. A fürt virtuális IP-címének pingelésével követni tudtam, hogy a szívverések kimaradásának észlelésekor, 4-5 másodperc elteltével a virtuális IP-cím megszűnt válaszolni, majd az átkonfigurálásakor újra elérhető lett. A hiba észlelése tehát a dokumentációnak [22] megfelelően zajlik (a szívverések elküldése 1,2 másodpercenként történik, három kimaradt szívverés után érzékel hibát a fürtszoftver), ám a hibaészlelés után csak több mint fél perces türelmi idő után kezdődik meg az újrakonfigurálás.

4.4 Quorumalapú fürt Windows rendszerekkel

4.4.1 Technikai részletek az előzetes tervezéshez

A quorumalapú fürtök fő jellegzetessége a quorumeszköz megléte. A quorumeszköz egyrészt praktikus megoldás a tudathasadás vagy az amnézia megelőzésére, másrészt tesztkörnyezetben messze nem triviális, hogy milyen módon biztosítható, hiszen egy tárolóhálózat beszerzése és beüzemelése pusztán tesztelési célból túlságosan drága lenne.

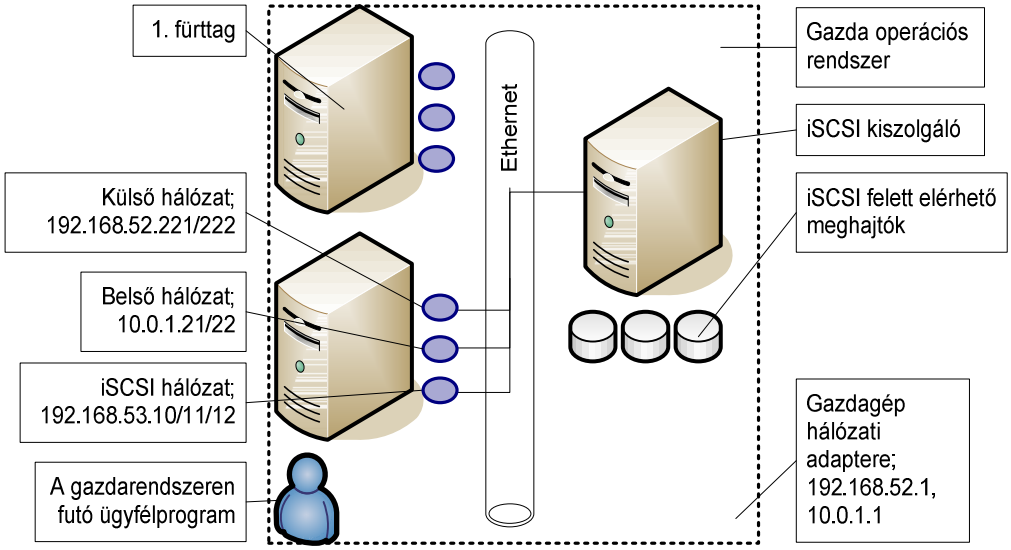
A quorumeszköz közös elérésű lemez, és mint ilyen, egyben az alkalmazások adatainak tárolására is használható. Ha az alkalmas adattároló rendszer rendelkezésre áll, akkor valószínűleg több közös elérésű lemez is hozzáférhető. Mivel az alkalmazások erőforrásfája sok esetben az alkalmazásadatokat tároló közös elérésű lemezre alapul, ha több közös elérésű lemezt is használni tudunk, akkor rugalmasabban alakíthatjuk ki az erőforráscsoportokat.

A fürtözni kívánt alkalmazásnak két feltételt kell teljesítenie:

- TCP/IP protokollt kell használnia a hálózati kommunikációra. A fürtszoftver csak IP-címeket tud erőforrásként kezelni.
- Az alkalmazásban konfigurálhatónak kell lennie az adattárolás helyének. Ennek hiányában nincs lehetőség arra, hogy az adatokat a közös elérésű lemezen helyezzük el, és azokhoz feladatátvételkor a másik fürttag is hozzá tudjon férni.

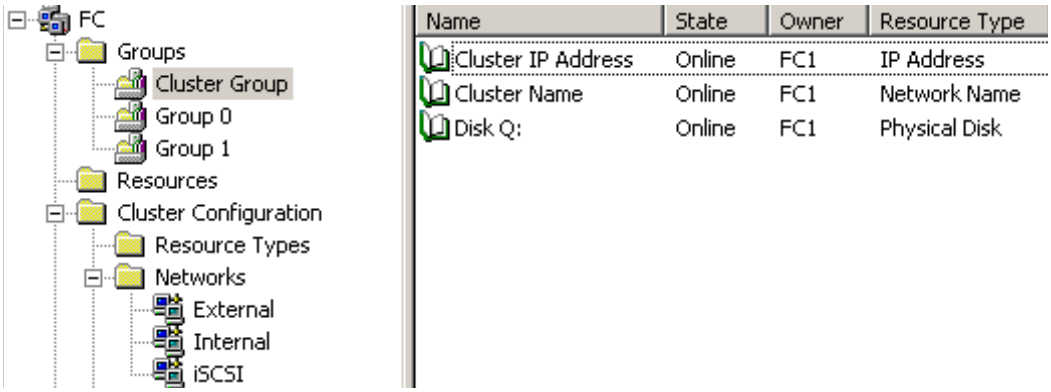
4.4.2 Tesztkörnyezet

A korábbi fürtökhöz képest ebben az esetben egy további virtuális géppel bővítettem a konfigurációt. A harmadik gép feladata a közös elérésű lemezek biztosítása volt, ezeket iSCSI protokollon keresztül érte el a két fürttag. A konfiguráció három közös elérésű lemezt tartalmazott, így a quorum tárolása mellett az egyéb alkalmazásadatoknak a quorumtól független kezelésére is módom volt. A fürtszoftver a közös elérésű lemezeket SAN-on, tárolóhálózaton keresztül elérhető lemezként látta. A fürttagokra a Microsoft iSCSI Initiator segédprogramot telepítettem. A fürttagokat a virtuális SAN elérése céljából egy harmadik hálózati adapterrel bővítettem. Az iSCSI-kiszolgálót előre konfigurálva, az internetről töltöttem le [41], az iSCSI Enterprise Target kiszolgálószoftver futott rajta [42].



31. ábra: Tesztkörnyezet quorumalapú fürt építéséhez

Az ábrán szereplő struktúra összeállítása után a fürt telepítése már egyszerűen, varázsló segítségével végezhető el. (Az egyszerűség kedvéért az ábrán nincs minden elem felcímkézve, és nincs minden hálózati kapcsolat jelölve.) A telepítés után az alábbi ábrán látható konfiguráció jött létre.



32. ábra: A quorumalapú fürt konfigurációja

A fürtszoftver három közös elérésű lemezt talált. Annak megfelelően, hogy a gyakorlatias alkalmazásokhoz olyan adattároló kell, amelynek tartalma alapján a tartalék fürttag is folytatni tudja a szolgáltatás működtetését, valamint a feladatátvétel egysége az erőforráscsoport, három erőforrás jött létre; mindegyikbe egy-egy lemez került. Az első ilyen csoport a fürtcsoport, a benne lévő Q: lemezen tárolódik a quorum. A Group 0 és a Group 1 csoport szabadon használható szolgáltatások konfigurálására.

Az alapszintű teszteléshez egy fájlmegosztást hoztam létre, ennek során a 14. ábra függőségi fáját építettem fel.

4.4.3 Mérések

Elsőként arra voltam kíváncsi, hogy mennyi ideig tart a Cluster Group felügyeleti célú áthelyezése egyik fürttagról a másikra. A mérésem szerint a quorum és a fürt virtuális IP-címének átadása, tehát offline állapotba állítása, majd a másik fürttagon online állapotba hozatala mindössze három másodpercig tartott, amelyet követően további 7 másodpercet igényelt a fürt nevének online állapotba kerülése. Ha eközben folyamatosan pingeltem a korábban említett paranccsal a fürt virtuális IP-címét, akkor mindössze egy válasz maradt ki.

Hasonlóan rövid ideig tartott a szívverések továbbítására szolgáló hálózati kapcsolat megszakadásának észlelése. 8 másodperccel a hálózati csatoló letiltása után a fürtszoftver letiltotta a második fürttagot. (A művelet megkezdésekor a quorumot az 1. fürttag kezelte.)

Ezt követően az első fürttagnak az iSCSI kiszolgáló elérésére használt hálózati kapcsolatát szakítottam meg. Kicsivel több, mint egy percig tartott, mire a Windows késleltetett írási hibát jelzett, majd 1 perc és 45 másodperc elteltével már a második fürttagon volt online állapotban a Cluster Group összes erőforrása. Mindeközben 28 másodpercig nem válaszolt a fürt virtuális IP-címe.

Hasonló időket mértem a fájlmegosztás erőforráscsoportjánál is. A két fürttag közötti felügyeleti célú áthelyezés 10 másodpercig tartott, a folyamat itt is a hálózati név miatt tartott ilyen „sokáig”, a közös elérésű lemez és az IP-cím áthelyezése kézzel nehezen mérhető, talán egy másodpercnyi időt igényelt.

A fájlmegosztás folyamatos elérhetőségét ügyféloldalról is teszteltem. A megosztáson belül létrehoztam egy szövegfájlt, megnyitottam a gazdagépről, és írtam bele valamit. A fájl mentése előtt a fürtszoftverrel áthelyeztem az erőforráscsoportot a másik fürttagra, és megpróbáltam menteni a fájlt. Attól számítva, hogy az erőforráscsoport minden erőforrása online állapotba került, 10-15 másodpercet kellett arra várnom, hogy a fájl mentésére tett kísérlet hatására ne jelezzen hibát a rendszer. Ha tehát a felhasználói dokumentumok tárolására fürtözött fájlmegosztást használunk, akkor ügyféloldalról mindössze az áthelyezés és a várakozási idő összegével, legfeljebb 25 másodpercnyi kieséssel kell számolnunk, amelyet a felhasználók nagy valószínűséggel nem is érzékelnek, hacsak nem éppen ekkor próbálnak dokumentumot elérni vagy menteni.

4.5 A teszteredmények összegzése

A mérési eredményeket a következő táblázatban foglaltam össze. Ismét ki szeretném emelni, hogy a mérések nem éles környezetben, hanem virtuális gépekkel készültek, ezért csak nagyságrendi iránymutatásként alkalmazhatók.

1. táblázat: A mérési eredmények összegzése

Fürttípus	Felügyeleti módosítások végrehajtási ideje	Hibakezelés időigénye
NLB	Nem észlelhető	Nem észlelhető
MNS	1-16 mp	45 mp
Quorumalapú	10 mp	105 mp

A hibakezelés időigénye egyben a szolgáltatáskiesés idejére is felső határértéket jelent.

5 A rendelkezésre állás üzleti és műszaki szempontból

5.1 Bevezető

Amint a korábbi fejezetekből kiderült, a rendelkezésre állás fokozására bőséges eszköztár áll a rendelkezésünkre, ám óvakodni kell attól, hogy a műszaki fejlesztések öncélúvá váljanak. Egy fürtözött rendszer kiépítése a szükséges hardverek, hálózati eszközök és szoftverek beszerzésével, a tanácsadói díjak kifizetésével és az esetleges oktatás díjával együtt bőven olyan tételt tesz ki, amelyet megalapozott számításokkal kell alátámasztani. A számok, a költségek és az elkerült károk összevetésének képessége a mindig maximalista elvárásokat megfogalmazó, ám a szükséges háttért ritkán biztosító üzleti szektorral vívott küzdelmekben is hasznos fegyvernek bizonyulhat.

A témakörrel kapcsolatos pontos definíciók ismerete hozzásegít ahhoz is, hogy a műszaki jellegűnek álcázott, valójában azonban marketingcéllal készült termékleírásokat kellő fenntartással tudjuk kezelni. Az ilyen leírásokban a cégek – el nem ítélhető módon, törekedve arra, hogy jobb színben tüntethessék fel a terméküket – sokszor helytelenül, félrevezető módon használják a kifejezéseket; az értékek ellenőrzésének képessége vagy a valós értékek nagyságrendi ismerete fontos segítséget jelenthet az ilyen dokumentációk kezelésében.

5.2 A rendelkezésre állás definíciója

A rendelkezésre állás formális meghatározásához a következő mutatókra van szükségünk [1]:

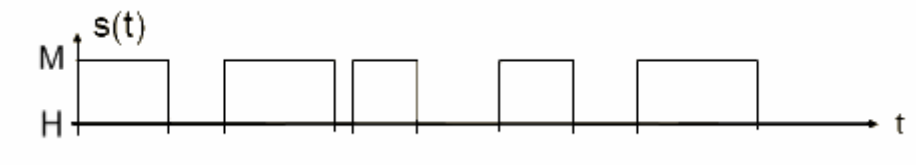
- *Megbízhatóság*: Annak a valószínűsége, hogy az adott *t* időpontig a vizsgált alkatrész, részegység vagy rendszer nem hibásodik meg, azaz:

$$r(t) = P\{t < T\} \ ,$$

ahol T az első hiba időpontja

- *MTTF* (Mean Time To Failure): A meghibásodásig átlagosan eltelt idő. Feltételezve, hogy a rendszer javítható, a javítást követően az időmérés újrakezdődik.
- *MTTR* (Mean Time To Repair): a rendszer javításához szükséges idő. A javítás a rendeltetésszerű működés helyreállítását jelenti. Az, hogy ez tényleges javítással vagy valamilyen alkatrész vagy részegység cseréjével valósul meg, lényegtelen.

Jelöljük a rendszer – időfüggő – állapotát s(t)-vel. Ha javítható a rendszer, akkor az életpályája a következő ábrával jellemezhető.

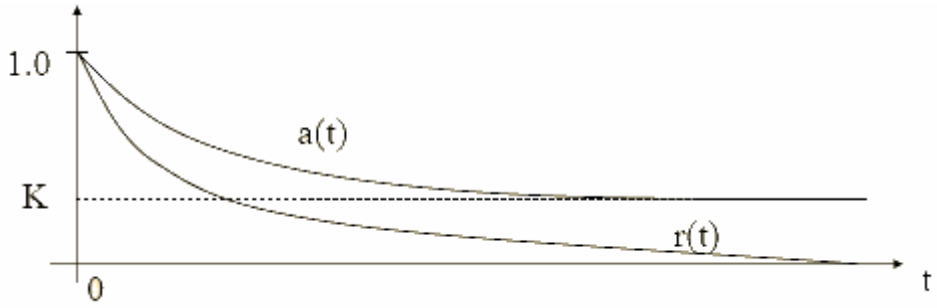


33. ábra: Egy rendszer élete

Ahogy az idő múlik, a megbízhatóságtól függően a rendszer valamikor meghibásodik, ekkor működő (M) állapotból hibás (H) állapotba kerül. A meghibásodáskor kezdetét veszi a javítás, amelyet követően a rendszer ismét működő állapotba kerül. Az idő bizonyos hányadát tehát működőképes, fennmaradó hányadát pedig működésképtelen állapotban tölti a rendszer. A rendelkezésre állás ezt az arányt, vagyis annak a valószínűségét fejezi ki, hogy a rendszer működőképes állapotban van:

$$a(t) = P\{s(t) \in M\}$$

Ha hosszú távon vizsgáljuk a rendszer életét, akkor a megbízhatósága tart a nullához, hiszen előbb-utóbb minden rendszer meghibásodik. A rendelkezésre állás az új – és még nagy valószínűséggel megbízható – rendszerek esetében értékének 100%-áról indul, majd csökkenni kezd, és a javításoknak köszönhetően idővel beáll egy 0 és 1 közötti értékre (0 és 100% közé), ezt nevezzük *készenléti tényezőnek* (K). Ezeket az értékeket szemlélteti az alábbi ábra.



34. ábra: A rendelkezésre állás, a megbízhatóság és a készenléti tényező időbeli alakulása

A gyakorlatban a rendelkezésre állás kevésbé érdekel bennünket, sokkal fontosabb a rendszerek üzemeltetésekor hosszú távon elérhető készenléti tényező. Ugyanakkor a különféle forrásokban rendkívül ritka a két érték megkülönböztetése, és a készenléti tényezőt szokták rendelkezésre állásnak nevezni – a továbbiakban én is átveszem ezt a szóhasználatot.

A készenléti tényezőt a szintén előszeretettel használt MTTF és MTTR alapján a következőképpen fejezhetjük ki:

$$K = \frac{MTTF}{MTTF + MTTR}$$

A műszaki szemléletet tükröző meghatározások mellett egy üzleti jellegű definíció is megfogalmazható [10]: a rendelkezésre állás azt fejezi ki, hogy egy alkalmazás vagy szolgáltatás elérhető-e, valamint képes-e a szükséges funkcionalitás biztosítására, amikor és amilyen mértékben azt a végfelhasználó igényli.

Látható, hogy a tisztán műszaki meghatározás még jellegében is más, mint az üzleti szemléletű. A következő szakaszban a rendelkezésre állás lehetséges „értelmezéseiről” is szó lesz.

5.3 Rendelkezésre állás a gyakorlatban

5.3.1 A megbízhatóság fokozása és a javítási idő szerepe

A termelési rendszerekben természetes igény a lehető legnagyobb rendelkezésre állás, illetve készenléti tényező elérése. Ennek a törekvésnek kézenfekvő része, hogy a megbízhatóság fokozásával növeljük az egyes meghibásodásokig eltelt időt, ám a javítási idő csökkentésének fontosságát sem szabad lebecsülni. A javítási idő szerepét a következő egyszerű számítással szemléltethetjük.

Tegyük fel, hogy a rendszerünk havonta egyszer hibásodik meg. Az egyik lehetőség, hogy a javításra külső szolgáltatóval kötünk szerződést, amely 4 óra alatt végzi el a javítást. A másik lehetőség, hogy magunk gondoskodunk a cserealkatrész tárolásáról és beszereléséről, és ezt a munkát fél óra alatt el tudjuk végezni.

2. táblázat: A javítási idő csökkentésének hatása

MTTF	MTTR	K
672	4	0,99408
672	0,5	0,99925

Látható, hogy a rendszer megbízhatóságát változatlanul hagyva, a javítási idő csökkentésével 99,4%-os helyett 99,92%-os rendelkezésre állást, vagyis számottevő javulást érhetünk el.

A megbízhatóság adott számítógép-kategóriát vagy operációs rendszert véve csak meghatározott keretek között befolyásolható, például a tapasztalatok szerint jó minőségűnek bizonyuló hardverelemek vásárlásával vagy a felügyeleti eljárások pontos szabályozásával. A javítási idő ellenben nagyságrendileg is javítható például tartalék alkatrész raktáron tartásával vagy kellően rövid javítási időt garantáló szolgáltatási szerződés megkötésével, és a fenti példa alapján érdemes lehet áldozni erre.

5.3.2 Célértékek

A gyakorlati életben a rendelkezésre állást százalékosan szokás megadni, és bevált megoldás a kilencesek számával kifejezni; a 99,99%-os rendelkezésre állást „négykilences” értéknek nevezik. A létfontosságú rendszerek esetében jellemzően az ötkilences, 99,999%-os rendelkezésre állást célozzák meg.

Az alábbi táblázat rövid áttekintést nyújt arról, hogy az egyes százalékos értékek mennyi leállási időt jelentenek éves, havi és napi szinten [3].

3. táblázat: A százalékos rendelkezésre állási értékek értelmezése

Rendelkezésre állás százalékos értéke	Leállási idő napi szinten	Leállási idő havi szinten	Leállási idő éves szinten
99%	14,4 perc	7 óra	3,65 nap
99,9%	86,4 másodperc	43 perc	8,77 óra
99,99%	8,64 másodperc	4 perc	52,6 perc
99,999%	0,86 másodperc	26 másodperc	5,26 perc

Van néhány olyan irányadó tapasztalati érték, amelyek alapján tudhatjuk, hogy egy-egy rendszertípustól hosszú távon milyen értékeket várhatunk. Természetesen az egyedi rendszerek rendelkezésre állása ettől akár nagyságrendileg is eltérhet, bármilyen irányban, ám kiterjedt statisztika készítésekor az alábbi értékekre lehet számítani:

4. táblázat: Nagyságrendi várható rendelkezésre állási értékek

Kiépítés	Célérték
Egy kiszolgáló	99,9%
Feladatátvételi fürt	99,99%
Feladatátvételi fürt felső kategóriás hardverrel, hibatűrő adattárolással	99,999%

Látható, hogy teljesen átlagos megoldást, egyetlen kiszolgálót véve is viszonylag jó eredményre számíthatunk, a nélkülözhető szolgáltatások esetében tehát érdemes lehet a költséghatékonyság irányába fordulni. Ha újabb kilencesekkel akarjuk javítani a rendelkezésre állást, akkor fürtözésre és egyre komolyabb hardver- és szoftvereszközökre kell forrásokat fordítanunk.

A rendelkezésre állás fokozása tehát költséges, minden esetben kellő gazdasági számításokkal kell alátámasztani a fejlesztés létjogosultságát, illetve műszaki-gazdasági kompromisszumra kell törekedni. Itt is érvényes, hogy a zökkenőmentesen üzemelő szolgáltatást mindenki természetesnek veszi, a rá fordított források akár elveszítettnek is tűnhetnek. A nagy rendelkezésre állás közvetlen hasznot nem hoz, közvetett haszna az elmaradt kiesések miatt nem jelentkező veszteségek révén mutatkozik meg. Azt, hogy milyen szintű rendelkezésre állást érdemes megcélozni, és melyek azok a szolgáltatások, amelyek valóban nélkülözhetetlenek a szervezet számára, az adott – folyamatosan változó – versenykörnyezet, a vetélytársak szolgáltatásai és árai, az adott szolgáltatás piaci pozicionálása is befolyásolhatják.

A rendelkezésre állás tervezésénél vagy meglévő rendszer mutatóinak javításakor mindig a teljes rendszert, a teljes környezetet kell vizsgálni [4]. Hiába javítjuk például a kiszolgálói hardverkörnyezet minőségét, ha az ügyfelek hozzáférését biztosító internetkapcsolatnak rosszak a minőségi értékei.

A rendelkezésre állás fokozásának objektív akadályai is lehetnek. A Microsoft Corporation által publikált [3] útmutatóban 3 és 5 perces feladatátvételi időket említenek, mint működő rendszeren mért értéket. Mint a 3. táblázat tartalmából kitűnik, 5 és fél perces kieséssel az ötkilences rendelkezésre állás elérése már lehetetlenné válik. Ha tehát egy fürt éves szinten csak két feladatátvételt hajt végre, akkor a legjobb hardverkörnyezetben is „csak" nagyságrendileg ötkilences rendelkezésre állásra képes.

Támpontként érdekes lehet néhány ismert szolgáltatás rendelkezésre állása [17]. Az olyan nagy szolgáltatók, mint a NASDAQ vagy a Buy.com a gyakorlatban három- vagy négykilences rendelkezésre állást érnek el. A PanTel Kft. a telefonszolgáltatására és az ADSL internet-hozzáférésekre 98%-os rendelkezésre állást ad meg célkitűzésként; a gyakorlatban három- vagy négykilences értéket érnek el [40].

5.3.3 A rendelkezésre állás, mint szolgáltatásminőségi jellemző

A rendelkezésre állás definíciója meglehetősen kevés támpontot ad a mindennapi rendszerekre nézve. A legtöbb esetben kevés egy egyszerű „rendelkezésre állási" mértéket előírni, ennél pontosabb szolgáltatási paramétereket kell kidolgozni. Előfordulhat, hogy a rendelkezésre állás mértékét nem annak alapján határozzuk meg, hogy az adott rendszer egyszerűen fut, üzemel-e, hanem aszerint, hogy valóban képes-e kiszolgálni a hozzá intézett kéréseket. Egészen más jellegű követelményt fogalmazhatunk meg azzal, hogy a szolgáltatásnak a kérések 99 százalékát kell kiszolgálnia, mint azzal, hogy a szolgáltatásnak az idő 99 százalékában kell működnie.

Lényeges kérdés a karbantartásokhoz szükséges idő kezelése. A szolgáltató számára kedvezőbb megoldás, ha a bejelentett, előre ütemezett karbantartásokat szolgáltatáskiesés mellett végezheti. A szolgáltatást igénybe vevő számára viszont érdektelen lehet a kiesés jellege, és a szolgáltatóra bízhatja, hogy hogyan oldja meg a karbantartások végrehajtását – logikusan az ügyfél szolgáltatást vásárol, nem foglalkozik a szolgáltató rendszerének belső felépítésével.

5.3.3.1 A rendelkezésre állás, mint időbeli elérhetőség

Ha időbeli követelményeket írunk elő, akkor a legegyszerűbb eset azt elvárni, hogy a teljes idő adott hányadában működjön a szolgáltatás. Előfordulhat azonban, hogy egy szolgáltatásra csak munkaidőben van szükség, és az esetleges éjszakai leállások ideje érdektelen. Ha 8 órás munkaidővel és öt munkanappal számolunk, akkor egy olyan rendszertől, amelynek az elérhetőségét csak munkaidőben vizsgáljuk, a teljes időre vetítve mindössze 23,8%-os rendelkezésre állást várunk, ami első megközelítésben katasztrofálisan rossznak számítana. Sok esetben a leállások jellege sem érdektelen; ha egy épületben óránként 2 másodpercre kimarad az áramellátás, akkor az időbeli rendelkezésre állás ugyan magas, ám a berendezések állandó újraindulása miatt akár lehetetlenné válhat a munkavégzés. Alkalmazási területtől függ, hogy a százalékos rendelkezésre állással megengedett kiesés több rövidebb vagy kevesebb, de hosszabban

tartó kimaradás formájában „előnyösebb” a felhasználó számára. Egy fájlkiszolgáló rövid idejű kimaradásait a felhasználók talán nem is észlelik, míg az áramellátás esetében a hosszabb idejű kimaradások okoznak kevesebb problémát.

5.3.3.2 A rendelkezésre állás, mint válaszadó képesség

Ha a rendszer elérhetőségét úgy definiáljuk, hogy képesnek kell lennie a beérkező kérések kiszolgálására, akkor is többféle lehetőséggel kell számolnunk.

Az első kérdés, hogy a működőképesség az összes funkció elérhetőségét jelenti-e; például egy adatbázis-kiszolgáló esetében a lekérdezési lehetőség már elegendő lehet az üzleti funkciók támogatásához, akkor is, ha a felügyeleti célt szolgáló táblalétrehozás éppen nem működik.

A második kérdés, hogy mit tekintünk a kérések kiszolgálásának, a kapcsolat létrejöttét, vagy éppen az alkalmazási szintű kérés befogadását, teljes és eredményes feldolgozását? Hiába jönnek létre a hálózati szintű kapcsolatok, ha a tranzakciók alkalmazási szintű kezelése visszaléptetéssel zárul, a hosszan futó, kötegelt feldolgozások megszakadnak, esetleg a feldolgozás sikeres ugyan, ám az időtartama a felhasználók számára elfogadhatatlanul hosszú. Ilyenkor a szolgáltatás működik ugyan, lényegi feladatát mégsem látja el.

A harmadik probléma, hogy a kérések érkezési intenzitása a különböző időszakokban eltérő lehet, ezért csúcsterhelésnél adott idejű kiesés jóval több kérés elveszítését jelenti, mint a kisebb forgalmú időszakokban; erre a forgalom elemzésével lehet csak felkészülni.

5.3.4 A rendelkezésre állás és a skálázhatóság viszonya

A rendelkezésre állás tervezése szoros kapcsolatban állhat a teljesítmény méretezésével. Egyrészt fürtözéssel részben a skálázás is kezelhető, másrészt figyelembe kell venni a rendszerek egymásra hatását.

A terheléselosztási megoldások egyszerű megoldást adhatnak a horizontális skálázás problémájára – egy-egy újabb fürttag üzembe állításával könnyen bővíthető a kiszolgálói kapacitás, illetve a csúcsidőszakon kívül könnyen csökkenthető is.

A feladatátvételi fürtök esetében – mivel minden szolgáltatás csak egy-egy fürttagon fut – horizontális skálázásra nincs lehetőség, a skálázást vertikálisan kell megoldani; ehhez viszont már kezdetkor is megfelelő hardverelemeket kell választani.

Figyelembe kell venni, hogy egy elvileg hibátlanul működő, ám túlterhelt szolgáltatás nem tudja kiszolgálni az ügyfeleket. Összetett munkafolyamatoknál egy-egy láncszem gyengesége miatt a munkafolyamat tagjainál feltorlódhatnak a feladatok, ami túlterhelődéshez, időtúllépéshez és egyéb problémákhoz vezethet. Hasonló méretezési kérdések a feladatátvételi topológiák kapcsán is felmerülhetnek, ezekre a topológiák rövid ismertetésekor már utaltam.

Fürtök segítségével viszonylag összetettebb, intelligensebb megoldásokat is kidolgozhatunk.

- Ha moduláris alkalmazást futtatunk, akkor a különböző modulokat egy fürt különböző gépeire oszthatjuk szét, így hatékonyan skálázhatjuk a teljesítményt. Ha a terhelés fokozatosan növekedik, akkor költséghatékony eljárás lehet, ha először csak kevés részre bontjuk az alkalmazást, majd az igények bővülésével egyre többfelé osztjuk, míg végül akár mindegyik modulhoz külön gépet állíthatunk üzembe. Ha a fürttagok között a feladatátvétel lehetősége is biztosított, akkor egyben az alkalmazás rendelkezésre állását is növeltük, azzal a kompromisszummal, hogy meghibásodás esetén nagy valószínűséggel túlterheltté válik legalább egy fürttag, ami az alkalmazás egy részének vagy egészének lelassulását fogja okozni.
- Hasonló megoldás az, amikor egyetlen tagból álló fürtöt építünk. Ekkor a virtuális kiszolgálók létrehozása egyrészt megkönnyíti a felügyeletet, másrészt, ha a kezdetben az összes szolgáltatást egymaga futtató számítógép teljesítménye a későbbiek során kevésnek bizonyul, akkor a fürthöz csak újabb tagokat kell csatlakoztatnunk, a szolgáltatások átkapcsolásával zökkenőmentesen bővíthetjük a kiszolgálóoldali rendszert.
- Ha több kisebb alkalmazást futtatunk, amelyek terhelése dinamikusan, de egymással ellentétesen változik, akkor fürtözéssel erőforrásokat takaríthatunk meg. Praktikus megoldás lehet egy nagyobb és több kisebb teljesítményű számítógépet fürtözni, és mindig azt az alkalmazást kapcsolni át a nagy teljesítményű gépre, amely iránt éppen a legnagyobb az igény. Ez a megoldás egyszerű felügyeletet tesz lehetővé, felesleges kapacitások hiányában takarékosnak számít, ám meghibásodás esetén itt is teljesítményproblémák jelentkezhetnek.

A teljesítmény skálázása kapcsán fontos megemlíteni, hogy az összes HA fürt esetében problémát jelenthet az adattároló alrendszer korlátozott teljesítménye. Míg a fürt bővíthető és a fürttagok viszonylag könnyen kicserélhetők, az adatok nagyobb teljesítményű tárolórendszerre való áthelyezése komolyabb felkészültséget igényel, ezért a kezdeti méretezésére is nagyobb figyelmet kell fordítani.

6 A rendelkezésre állás elemzése

6.1 Bevezető

Bár a különféle rendszerekhez hozzá lehet rendelni bizonyos tapasztalati adatokat (lásd: 4. táblázat), előfordulhat, hogy egy nagyobb költségű, például a rendelkezésre állás négykilencesről ötkilencesre való javítását célzó beruházás jogosságát formális módszerekkel is érdemes vagy kell igazolni. A vizsgálathoz készíteni kell a rendszerről valamilyen formális modellt, ezt el kell látni a megfelelő, a valóságot a lehető legjobban tükröző paraméterekkel, majd le kell futtatni a szükséges vizsgálatokat vagy szimulációkat. Az eredménytől függ, hogy a szükséges ráfordítások tükrében vajon érdemes-e egyáltalán megkezdeni a fejlesztést. A nagyobb beruházások esetében érdemes számolni azzal is, hogy a modellezés viszonylag olcsón elvégezhető, és egyszerűségét tekintve is maga mögé utasítja például a valós eszközökkel végzett méréseket.

Mielőtt megkezdhetnénk bármilyen modell felállítását, fel kell mérnünk, hogy milyen tényezők befolyásolhatják a rendelkezésre állást. Ennek lépései a következők:

1. A lehetséges meghibásodások feltérképezése, majd a különféle hibák elhárítási módjának felmérése.
2. A megbízhatósági adatok beszerzése, amely tapasztalati adatok összegyűjtésével vagy a gyártói adatok bekérésével valósulhat meg.
3. A rendszerelemek egymáshoz fűződő viszonyának áttekintése valamilyen egyszerű eszközzel, például hibafákkal.
4. Modellalkotás, majd a modell alapján történő elemzés és érzékenységvizsgálat.

A következő szakaszokban ezeken a pontokon fogok végighaladni.

6.2 A meghibásodási okok kategorizálása

Ha adott rendszer rendelkezésre állásának javítására törekszünk, akkor meg kell vizsgálnunk, hogy a leállások milyen okokra vezethetők vissza. Négy alapkategóriát minden esetben érdemes elkülöníteni: a hardverhibákat, a szoftverhibákat, az emberi tevékenységre visszavezethető leállásokat és a környezeti hatásokat. Alkalmazási területtől függően az alapkategóriák a következőképpen bonthatók tovább [18] [19] [20] [21]:

- Hardverhibák
  - alaprendszer (alaplap, processzor, memória)
  - tápellátás (tápegység, szünetmentes táp, áramátkapcsoló egységek)
  - adattároló alrendszer
  - hálózat

- Szoftverhibák
  - az operációs rendszer hibái
  - alkalmazáshibák
  - illesztőprogram-hibák
- Emberi hibák
  - rendszergazdai hibák
  - illetékes felhasználók nem rosszindulatú hibái
  - illetékes felhasználók rosszindulatú hibái, támadásai
  - illetéktelen felhasználók támadásai
- Környezeti hatások
  - üzemeltetési környezet rendellenességei, például a légkondicionálás leállása, bombariadó, csőtörés
  - természeti katasztrófák

Mindig a tényleges rendszertől függ, hogy mely kategóriákat érdemes figyelembe venni. Egy kizárólag belső hálózaton át elérhető fájlkiszolgáló esetében vélhetően nagyobb szerephez jutnak az adattároló alrendszer hibái, míg egy webkiszolgáló esetében a támadások arányának növekedésére kell számítani.

A vizsgált rendszertől függően további felbontás is elképzelhető. Ha a rendszer működésében nagy szerep jut a hálózatnak, akkor szükség lehet például a hálózati kapcsoló vagy útválasztó hardveres és szoftveres hibáinak elkülönítésére. Sok helyen a tervezett és a nem tervezett leállásokat is elkülönítik, itt azonban a nem tervezett leállásokkal, a hibákkal foglalkozok elsősorban.

A fenti kategorizáláson túl szükség lehet a hibák javításához szükséges idő vizsgálatára, illetve a javítási idő szerinti további felbontásra is. A támadások felosztásakor például nem lehet azonosként kezelni az operációs rendszer lefagyásához vezető, egy újraindítással és a szükséges felügyeleti műveletek elvégzésével rövid idő alatt javítható hibákat és azokat, amelyek a működéshez szükséges adatok sérülésével vagy megsemmisülésével járnak.

A kategorizálás mellett érdemes a hibák súlyosságát, hatásának kiterjedését és az általuk okozott kárt is összegezni, majd kiválasztani az összességében legsúlyosabbnak bizonyuló hibákat [5]. (Lásd: A hibamódok, a hibahatások és a hibák kritikusságának vizsgálata; failure modes, effects and criticality analysis, FMECA módszer)

Összetett szolgáltatás esetében – a fentihez hasonló módszerrel – érdemes lehet magát a szolgáltatást is több részre bontani. Egy webáruház esetében a böngészési funkció kimaradása kisebb kárral jár, mint a rendelési folyamat vagy a fizetési funkció kiesése, hiszen az utóbbi esetben a felhasználó már tényleges ügyféllé, vásárlóvá lépett elő.

6.3 Adatgyűjtés

A meghibásodási okok kategorizálása után az egyes hibaokokhoz hozzá kell rendelni valamilyen bekövetkezési gyakoriságot, valamint – figyelembe véve a javítási idő szerepét – fel kell mérni a javítások várható időigényét is. A kiinduló megbízhatósági adatok összegyűjtésére többféle módszert is választhatunk.

6.3.1 Microsoft Reliability Analysis Service (MRAS)

A Microsoft termékekre alapuló környezetek üzemeltetési adatainak összegyűjtésére praktikus eszköz a Microsoft Reliability Analysis Service. Az ingyenesen hozzáférhető szolgáltatás igénybe vételéhez mindössze egy ügyfélprogramot kell telepíteni, amely kigyűjti a rendszer eseménynaplójába készített bejegyzéseket, majd feltölti az adatokat egy webes szolgáltatásnak. A szolgáltatás ezek alapján különféle statisztikákat készít. Az egyes szervezetek által összegyűjtött adatok és a statisztikák sajnos bizalmasak, külső fél számára nem érhetők el, így előzetes tervezési célra nem használhatók. Ha azonban meglévő rendszert szeretnénk bővíteni, esetleg már vannak működő fürtjeink, akkor értékes tapasztalati adatokat gyűjthetünk össze vele.

6.3.2 Az eseménynapló elemzése

A Windows operációs rendszerek eseménynaplója minden rendszereseményhez (normál és rendellenes leállás, rendszerindítás stb.) rendel egy azonosító számot. Ha exportáljuk az eseménynapló tartalmát, akkor az azonosítók alapján statisztikát készíthetünk a rendszereseményekről.

Általánosabb megoldás a syslog, amellyel más operációs rendszerekre és az intelligensebb hálózati eszközökre is kiterjeszthetjük a naplózást. A syslog kiszolgáló üzembe helyezése egyszerű, a figyelni kívánt eszközökön pedig elég megadni naplózási helyként a kiszolgáló címét.

Mind a windowsos eseménynapló, mind a syslog kiszolgáló által összegyűjtött események elemzésére számtalan alkalmazás található az interneten, ingyenes és kereskedelmi konstrukcióban egyaránt. Sajnos ez a módszer is csak utólagos összesítések készítésére alkalmas, előzetes kiinduló adatokat máshonnan kell szereznünk.

A saját adatgyűjtésnek megvan az a kétségtelen előnye, hogy a saját környezetre jellemző információkat szolgáltat. Hiába találunk az interneten statisztikákat, a háttér-információk ismerete nélkül csak komoly fenntartásokkal kezelhetjük őket. Önmagában az, hogy egy kiszolgálót hetente újra kellett indítani a vizsgált időszakban, semmit nem mond. Ennek oka lehet az, hogy a szoftver gyártója hetente kiadott egy újraindítást igénylő frissítést, amit a gondos rendszergazda telepített is, de lehet az, hogy a kiszolgálót hetente megtörték, és a rendszergazda a biztonsági rés megszüntetése helyett mindig csak visszaállítja a korábbi állapotot.

6.3.3 Tanulmányok, mint adatforrások

Értékes információforrást jelenthetnek az interneten fellelhető megbízhatósági tanulmányok. Használatukat sajnos több tényező is nehezíti: egyrészt kevés az olyan tanulmány, amelyben tényleges számadatok is szerepelnek, másrészt sok tanulmány mára elavult, harmadrészt a tanulmányok készítői a legtöbbször gyökeresen eltérő kategorizálást és szemléletet követnek, így nagyon nehéz a fellelt anyagok alapján valamilyen egységes, több forrásból is megerősített képet kialakítani.

Az általam áttekintett források [18] [19] [20] [21] alapján a következő nagyságrendi adatokkal érdemes dolgozni. Szeretném kiemelni, hogy számértékeket még a legmerészebb tanulmányok is csak közelítésképpen említenek, így az alábbi táblázat is hangsúlyozottan csak távoli becslést adhat.

5. táblázat: Közelítő értékek a hibák gyakoriságára és javítási idejére

Hiba jellege	MTBF	MTTR
Hardver	8000 óra (kb. 1 év)	2 óra
Szoftver	360 óra (15 nap)	0,1 óra
Egyéb	720 óra (30 nap)	0,5 óra

A fentinél jóval részletesebben is feloszthatnánk a hibafajtákat, pontosabb képet alkotva a rendszer jellemzőiről, ám ezzel a modellezést is bonyolultabbá tennénk. Egészen más meghibásodási gyakoriságra kell számítani egy kiforrott adatbázis-kiszolgáló és például egy bonyolult, folyamatosan új és még csak minimális mértékben tesztelt technológiákra alapuló alkalmazáskiszolgáló esetében.

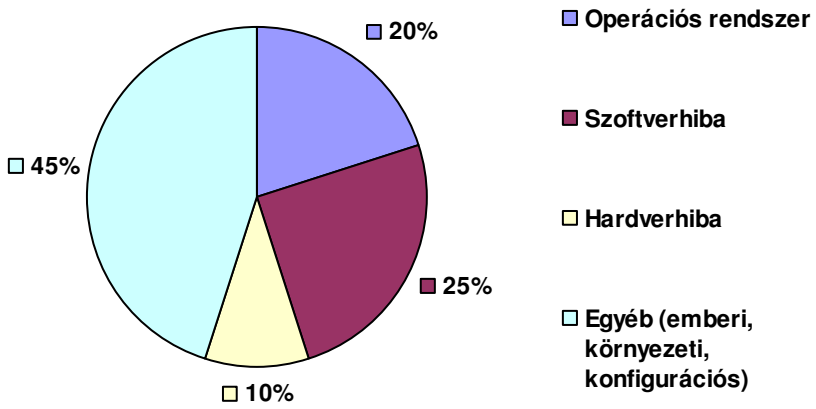
Nem kevésbé problémás a hibaokok eloszlásának becslése. Az említett tanulmányok hiányosságai kiválóan szemléltetnek néhány problémát:

- [18] esetében a 3. táblázatnak a maximális leállási időket tartalmazó sorában a Windows NT rendszereknél 2,3, a Windows 2000 rendszereknél pedig 1,3 hónapos idő szerepel maximális leállási időként. Nyilvánvaló, hogy egy kiszolgáló hónapos nagyságrendű leállása feleslegesen torzítja a statisztikát, hiszen nélkülözhető szolgáltatásról lehetett szó. Ez az aprónak tűnő hiányosság ugyanakkor rámutat arra a problémára, hogy adatforrásként csak kellő szakmai színvonalon üzemeltetett rendszert érdemes venni. Hol húzzuk meg azonban azt a határt, amelytől az üzemeltetés professzionális színvonalúnak számít? A tanulmány alapján egyértelműnek tűnik az is, hogy a vizsgálatok tárgyául nem érdemes akadémiai környezetet választani, ahol a felhasználók rendkívül sokféle és csekély mértékben szabályozott tevékenységet végeznek a munkaállomásokkal.
- [19] értékes, éles környezetből szerzett információkat szolgáltat (9. oldal), amely szerint a cég PRIMEPOWER gépei 1-2 évente hibásodnak meg. Bár a tanulmány szerzői – megítélésem szerint – alaposan körüljárták a témát, a 4. táblázatban

mégis egyaránt 8000 órás (nagyjából egy éves) MTBF értékkel számolnak a hardver, az operációs rendszer, az alkalmazások és a hálózat esetében egyaránt. A szerzők több helyütt is utalnak rá, hogy maguk is kénytelenek voltak becslésekkel, feltételezésekkel élni. Elgondolkodtató, hogy egy nagy gyártó mérnökei vajon rendelkeznek-e éles adatokkal, csak marketingszempontok miatt nem tehetik közzé őket, vagy maguk is csak becsülni tudnak.

- [20] 1. és 2. ábráján szembetűnő a sehova sem sorolt meghibásodási okok magas, 36 és 58 százalékos aránya. Utóbbi eredmény nehezen értékelhető, és egyben rámutat arra, hogy a hibaokok felderítésére és osztályozására megfelelően fel kell készülni az adatgyűjtés megkezdése előtt.
- [21] szerzői külön kiemelik, hogy a lehetséges hibaokok közül nem vették figyelembe többek közt az áramellátás és a hűtőrendszer meghibásodásait. Ha magas szintű rendelkezésre állásra törekszünk, akkor viszont ezeket sem hagyhatjuk ki a számításokból; ezzel azonban újabb adatforrásokat kell bevonnunk, illetve tovább bonyolódnak a számítások.

A talált források bizonytalansága miatt úgy döntöttem, hogy megpróbálom „összefésülni” a hibaokok eloszlását; ennek nyomán alakultak ki az alábbi arányok.



35. ábra: A hibaokok aránya

A felületes szemlélő hajlamos lehet arra, hogy elsősorban a technológiát, tehát a hardver- és szoftverelemeket okolja a szolgáltatáskiesésekért. A tapasztalat azonban azt mutatja – és talán ez az egyetlen pont, amelyben az összes forrásban egyetértés tapasztalható –, hogy egyrészt a hibák igen jelentős részét az emberek okozzák a technológia helytelen kezelésével, hibás konfigurálásával, másrészt folyamatosan csökken a technológiára visszavezethető hibaokok aránya. Azonnal levonható következtetés tehát, hogy a technológia javítására – például fürtözéssel – könnyen található megoldás, ám érdemes legalább ekkora figyelmet folytatni a rendszer felügyeleti eljárásainak kidolgozására, a felügyeletet ellátó személyzet képzésére, valamint a megfelelő dokumentáció elkészítésére és naprakészen tartására.

Az adatgyűjtést az MTBF és MTTR értékén túl további szempontokra is ki lehet terjeszteni, amennyiben ez szükséges; ezzel azonban túllépünk az egyszerű

statisztikakészítésen és az összefüggések feltárása felé mozdulunk el. Néhány ilyen szempont, a teljesség igénye nélkül:

- Az újraindítások csomósodásának figyelése. Sok esetben a felmerült hiba végleges elhárításához több újraindítás is szükséges.
- Előfordul, hogy egy-egy hiba adott hálózatrész összes számítógépén vagy valamelyik operációs rendszer minden példányán végigsöpör. Erre főként a biztonsági rések ismertté válása és a frissítések telepítése kapcsán kell figyelni.
- Értékes információval szolgálhat, hogy mi az egyes hibák javításához szükséges idő eloszlása. Ennek alapján meg lehet tenni a rendelkezésre állást fenyegető, hosszú javítási idők csökkentéséhez szükséges lépéseket.
- A hibák gyakorisága nem feltétlenül függ össze azzal, hogy az adott hibafajta mennyi leállási időért felelős. Ezért is fontos a javítási idő figyelése.

Általában elmondható, hogy a vizsgálatok a végtelenségig bonyolíthatók; az adott környezettől függ, hogy milyen összefüggéseket érdemes figyelembe venni.

A vizsgálatok során egyes műveletek időigényét pontosan meg lehet határozni, ilyen például egy meghibásodott tápegység cseréje, ha raktáron van a cserealkatrész. Más időtartamokat nehezebb mérni, például az operációs rendszer vagy a szolgáltatások újraindításának időigénye erősen függ attól, hogy a rendszernek mennyi függőben lévő műveletet kell lezárnia, milyen időtúllépéseket kell megvárnia. Megint más időket csak becsülni lehet, erre a legjobb példát a meghibásodások között eltelt idő jelenti, de említhető a sérült adatok helyreállításának idő- és munkaigénye is.

6.4 Hibafa

A hibafa egy meglehetősen alapszintű konstrukció, segítségével gyorsan és egyszerűen át lehet tekinteni, hogy a különféle meghibásodások milyen együttállások esetén vezetnek egy teljes rendszer meghibásodásához, valamint a hibafa alapján könnyedén azonosíthatók az egyszeres hibapontok. Az áttekinthetőség kedvéért érdemes tehát felrajzolni a fürtözött rendszerek hibafáját is.

6.4.1 Konstrukciós elemek

A hibafák felépítéséhez a következő elemeket fogom felhasználni. Ugyan a hibafák ennél többféle összetevőt is tartalmazhatnak, valamint az itt szereplőknél összetettebb, akár többszintű hibafák is készíthetők, a jelenlegi céljainknak ezek az alapeszközök is megfelelnek.

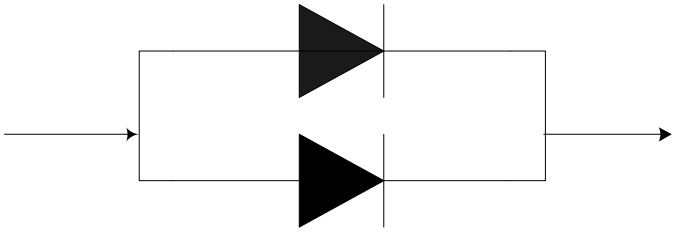
6. táblázat: A hibafák felépítésére használt elemek

Elem	Leírás
	Elemi esemény. Olyan meghibásodási eseményt jelez, amelyet nem részletezünk tovább.

	ÉS kapu. Elemi események vagy egyéb kapuk köthetők be alá, ezek bekövetkezte között ÉS logikai kapcsolatot létesít, tehát a kimenetén akkor ad IGAZ jelet, ha az összes bemenetén IGAZ bemenetet kap.
	VAGY kapu. Hasonló az ÉS kapuhoz, de a bemenetei között logikai VAGY kapcsolatot létesíti.

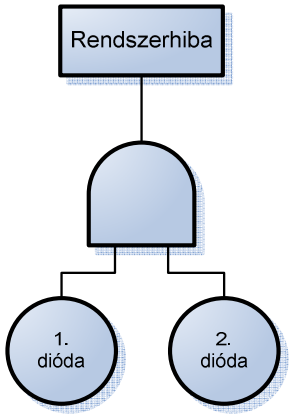
6.4.2 Alappélda

A hibafa használatának szemléltetésére vegyük a klasszikus példát, a két darab párhuzamosan csatolt diódát.



36. ábra: Két párhuzamosan kapcsolt dióda

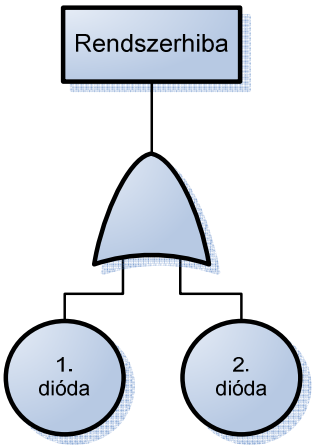
Tegyük fel, hogy az egyik dióda meghibásodik, és ennek során szakadás jön létre. Ebben az esetben a másik dióda még el tudja látni a feladatát, és végső soron a teljes kapcsolás működőképes marad. A rendszer működésképtelenné válásához tehát mindkét diódának meg kell hibásodnia. Ennek alapján a kapcsolás hibája a következő lesz.



37. ábra: Két diódából álló kapcsolás hibafája szakadási hibára

A fenti hibafa értelmében mindkét diódának meg kell hibásodnia ahhoz, hogy az ÉS kapun keresztül elérjünk a rendszerszintű hibához, és a kapcsolás ne tudja ellátni a funkcióját.

Ha azt feltételezzük, hogy a meghibásodó diódában rövidzár alakul ki, akkor a másik dióda hiába működne megfelelően, a kapcsolás rövidzárba ment át, és nem tudja ellátni a funkcióját. Ebben az esetben a hibafa a következő lesz.

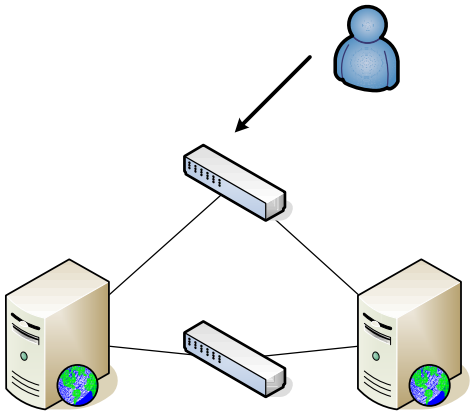


38. ábra: Két diódából álló kapcsolás hibafája rövidzár esetére

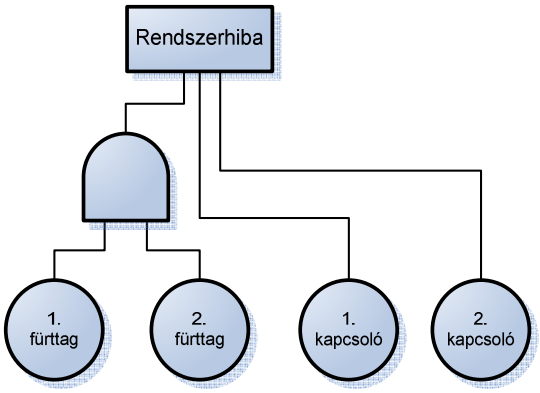
A hibafa struktúrája nem változott ugyan, ám az ÉS kapu helyét egy VAGY kapu vette át. Mivel itt egyetlen elemi hibaesemény is elegendő a rendszer leállásához, ennek a meghibásodási módnak a szempontjából a kapcsolás kevésbé megbízható.

6.4.3 Terheléselosztási fürt hibafája

Vegyük példaként az alábbi, a 4.2.2 pont szerinti, alapkiépítésű terheléselosztási fürtöt, amelynek rajzoljuk fel a hibafáját is.



39. ábra: Alapszintű terheléselosztási fürt felépítése

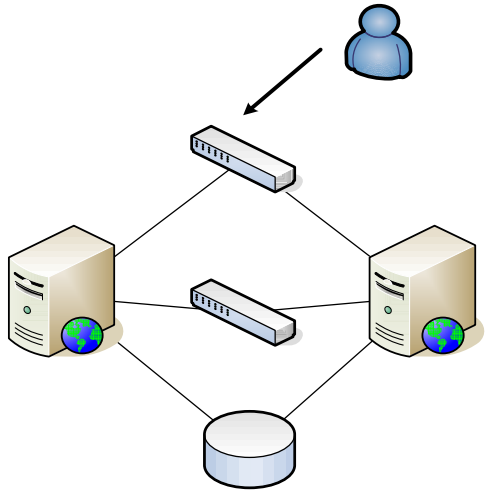


40. ábra: Az alapszintű terheléselosztási fürt hibafája

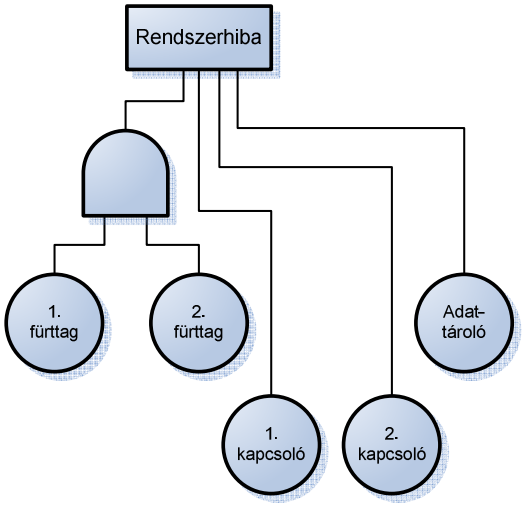
A hibafa alapján látható, hogy a rendszerhiba bekövetkeztéhez mindkét fürttagnak meg kell hibásodnia. Egyszeres hibapont ellenben, és közvetlenül rendszerszintű hibához vezet bármelyik hálózati kapcsoló hibája, hiszen a felső kapcsoló kiesésekor az ügyfelek hozzáférése, az alsó kiesésekor pedig a szívveréseknek a fürttagok közötti továbbítása hiúsul meg (feltételezzük, hogy a szívveréseket csak ezen a kapcsolaton keresztül tudják továbbítani a fürttagok). A fentinél jóval összetettebb, több rendszerelemre kiterjedő, akár a környezetet is magába foglaló hibafát is lehetne rajzolni, az alapvető áttekintéshez azonban a fenti egyszerű konstrukció is elegendő.

6.4.4 Feladatátvételi fürt hibafája

A fentihez hasonlóan rajzolhatjuk fel egy egyszerű, a 4.4.2 pontban is megépített, két számítógépből álló feladatátvételi fürt felépítését és hibafáját is. Tegyük fel, hogy a szívverések csak a fürt belső hálózatán továbbíthatók, és – azon kívül, hogy a fürt két tagból áll – még egyik rendszerelem megkettőzésére sem tettünk kísérletet.



41. ábra: Alapszintű feladatátvételi fürt felépítése



42. ábra: Az alapszintű feladatátvételi fürt hibafája

A feladatátvételi fürthöz képest változás, hogy új elemként megjelent egy újabb egyszeres hibapont, az adattároló rendszer.

Bár a hibafák alapján – a meghibásodási valószínűségek, az MTBF/MTTR értékek hozzárendelése után – számításokat is lehet végezni, ez esetben a cél csupán annyi, hogy át tudjuk tekinteni a rendszer struktúráját, és azonosítani tudjuk azokat a kritikus, egyszeres hibapontként megjelenő rendszerelemeket, amelyek egyébként esetleg elkerülhetnék a figyelmünket.

A hibafák egyik gyengéje, hogy mindig csak egy-egy meghibásodási módra terjednek ki, a különféle hibákhoz különböző hibafákat kell felrajzolnunk. Ha a hibák jellege hasonló, akkor ezek a hibafák azonos felépítésűek lesznek, csak a valószínűségi értékeik térnek el, ám ha közös módusú és független hibákat is figyelembe akarunk venni, akkor többféle fát is fel kell rajzolnunk.

6.5 Petri-hálók

6.5.1 Az exponenciális eloszlás

Mielőtt rátérnénk a Petri-hálókra, át kell tekintenünk az exponenciális eloszlást, ugyanis a továbbiakban szükségünk lesz rá [43].

Egy ξ folytonos valószínűségi változót exponenciális eloszlásúnak nevezünk, ha a sűrűségfüggvénye a következő:

$$f(x) = 0, \text{ ha } x \leq 0; \text{ és } \lambda e^{-\lambda x}, \text{ ha } x > 0,$$

ahol λ tetszőleges pozitív szám lehet, és amelyet az eloszlás paraméterének nevezünk.

Az exponenciális eloszlás érdekessége az örökifjú tulajdonság. Ha a ξ egy esemény bekövetkeztéig eltelő időtartamot jelöli, és ha a kiinduló időponttól vett tetszőleges T időpontig még nem következett be az esemény, akkor ez a T időpont is tekinthető a kiinduló időpontnak:

$$P(\xi \geq T+t \mid \xi \geq T) = P(\xi \geq t)$$

Az esemény bekövetkezésének valószínűsége tehát az idő múlásával nem nő.

Az exponenciális eloszlású, λ paraméterű valószínűségi változó várható értéke:

$$M(\xi) = 1/\lambda$$

Exponenciális valószínűséggel jellemezhető sok műszaki berendezés, elektronikai alkatrész meghibásodása, leszámítva élettartamának kezdeti, bejáratási és utolsó, elöregedett szakaszát. Ha például egy készülék várhatóan 1000 üzemóra után hibásodik meg, akkor $M(\xi)=1/\lambda=1000$, vagyis az alkatrész meghibásodásának valószínűsége a $\lambda=0{,}001$ paraméterű exponenciális eloszlással jellemezhető. A továbbiakban exponenciális eloszlással fogjuk leírni a fürtöket alkotó részegységek meghibásodásait is.

6.5.2 A Petri-hálók áttekintése

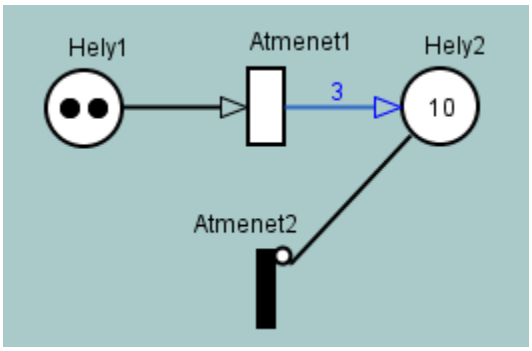
A Petri-háló egy viszonylag egyszerű konstrukció, amely ún. helyekkel és a helyek közötti átmenetekkel írja le egy rendszer viselkedését. Mivel a fürtök, illetve a fürtöket alkotó komponensek is állapotok – működő és hibás állapot – között mozognak, a Petri-hálókkal a fürtök viselkedését is modellezhetjük.

A Petri-hálóknak a különböző vizsgálatokhoz különféle változatait dolgozták ki. Ezek, valamint a Petri-hálókkal kapcsolatos fogalmak ismertetése nem célom; az alábbiakban csak a számomra fontos elemeket foglalom össze.

Az alapvető építőelemek a következők:

- **Hely**. A helyek a rendszer állapotait jelölik, itt „tartózkodhatnak” a tokenek.
- **Átmenet**. Az átmenet aktivizálásakor, *tüzelés*ekor az élek mentén a tokenek az egyik helyről egy másik helyre kerülnek át. Egy átmenet *engedélyezett*, ha elegendő token van a bemeneti helyén vagy helyein ahhoz, hogy tüzelhessen. Többféle átmenetet is megkülönböztetünk, a legegyszerűbb az *azonnali átmenet*, amely azonnal tüzel, ha engedélyezett. Egy átmenet *időzített*, ha nem azonnal tüzel, amint engedélyezetté válik, hanem a tüzelésig eltelő időt egy valószínűségi változóval adjuk meg. A továbbiakban főként exponenciális időzített átmeneteket fogok használni, amelyeknél a tüzelés idejét egy exponenciális eloszlású valószínűségi változó írja le.

- **Él**. Az élek határozzák meg, hogy milyen összeköttetések vannak a helyek és az átmenetek között. Az élekhez *súly* is rendelhető, az alapértelmezett súly egy. Az átmenet tüzelésekor az él súlyának megfelelő számú token változtat helyet.
- **Token**. A tokenek a helyeken elhelyezett és az átmenetek által áthelyezett egységek. A tokeneknek a helyeken való elhelyezkedése írja el, hogy a rendszer milyen állapotban van.



43. ábra: A Petri-hálók alapelemei

A fenti ábra ezeket az alapelemeket szemlélteti. Az ábrán két hely látható, a Hely1 és a Hely2. Előbbin két, utóbbin tíz token található. A Hely1 és a Hely2 hely között található az Atmenet1 időzített átmenet. Ha ez tüzel, akkor a Hely1 helyről egy tokent vesz el, a Hely2 helyre pedig hármat tesz be, mert az átmenet bemeneti éle egyes, a kimeneti éle pedig hármas súlyú. (Az átmenetbe bekerülő és a belőle kikerülő tokenek számának tehát nem muszáj egyeznie; ez egyes súlyt nem jelöljük külön.) Az Atmenet2 átmenet egy azonnali átmenet. A Hely2 és az Atmenet2 között egy tiltó él látható; ez azt fejezi ki, hogy az Atmenet2 nem tüzelhet, amíg a Hely2 helyen legalább a tiltó él súlyával egyező számú, a jelen esetben egy darab token található.

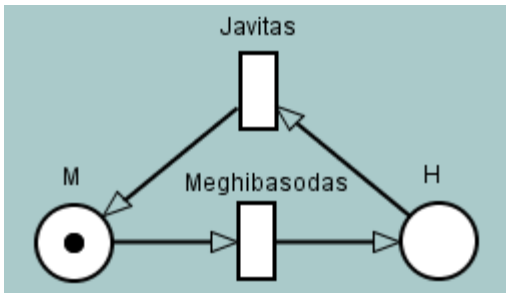
A Petri-hálók egy további kiegészítése a „*reward*” (díj, juttatás). Reward segítségével fejezhető ki például egy állapot „értéke”. Lehetőség van például olyan reward megadására, amelyet akkor kapunk, ha a fenti példa Hely1 helyén legalább egy token található.

(A Petri-hálókról az ábrákat a TimeNet eszközzel rajzoltam meg [45].)

6.5.3 Alapmodell

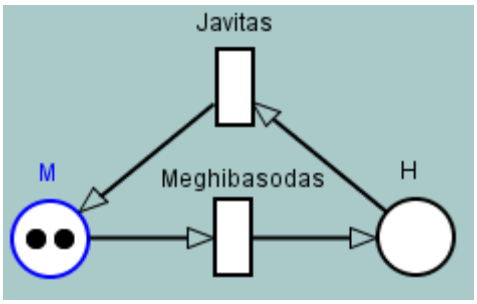
A fenti elemekből építkezve egy számítógép életét a 44. ábra szerint modellezhetjük.

A modell szerint a számítógép működő (M) állapotból egy exponenciális eloszlással jellemzett átmenettel hibás (H) állapotba kerül, vagyis meghibásodik. A hiba javítása szintén exponenciális eloszlással jellemezhető (hacsak a javítás nem egy előre megadott időtartam alatt elvégzett, teljes cserét jelent, ebben az esetben determinisztikus eloszlással és determinisztikus idejű átmenettel jellemezhetnénk), ezzel jut vissza működő állapotba a gép [1].



44. ábra: Egy számítógép életének modellje

Ha két gépünk van, és alapszintű modellezésre törekszünk, akkor mindössze annyit kell változtatnunk, hogy az M helyre két tokent teszünk. Ha pontosabb eredményt szeretnénk, akkor az átmenet tüzelési valószínűségét is módosítanunk kell, hiszen két számítógép közül az egyik nagyobb valószínűséggel hibásodik meg, mintha csak egy gépet vizsgálnánk. Erről a témáról lásd még a függeléket.



45. ábra: Kétgépes modell

A fenti modellt rewarddal is kiegészíthetjük. Ha például a cél az, hogy a két számítógépből legalább egy működjön, akkor a rewardot (R) kétféle módon definiálhatjuk:

- Ha a H helyen lévő tokenek száma kisebb 2-nél, akkor R=1,
- Ha az M helyen lévő tokenek száma nagyobb 0-nál, akkor R=1.

Ha a megfelelő paraméterekkel és szimulációs szoftverrel kiszámítjuk például a modell állandósult állapotbeli állapotát, valamint rewardként a fenti két R érték valamelyikének várható értékét definiáljuk, akkor megkapjuk, hogy mi a kétgépes rendszerünk készenléti tényezője.

6.5.4 Modellezési problémák

Kézenfekvő, hogy a fenti mintát követve, a fürt egyes összetevőit hasonló, két helyet és két átmenetet tartalmazó részekkel leírva tetszőleges számú összetevő sorsát tudjuk modellezni, és ezeket együtt kezelve maga a fürt is modellezhető. Ezt a megoldást fogom alkalmazni, ám előtte szeretnék kitérni néhány problémára.

- Mit tartalmazzon a modell és milyen részletességgel? A hibafáknál már utaltam rá, hogy a teljesség érdekében érdemes a környezetet (hűtés, áramellátás, épület) is belefoglalni a modellbe, azonban ezzel növeljük annak bonyolultságát, és további adatgyűjtési problémákkal szembesülünk. További kérdés, hogy a teljes

rendszer egyes elemeit milyen részletességgel szemléljük; például egy számítógépet osztatlan egységként kezelünk, vagy különvesszük a részegységeit.

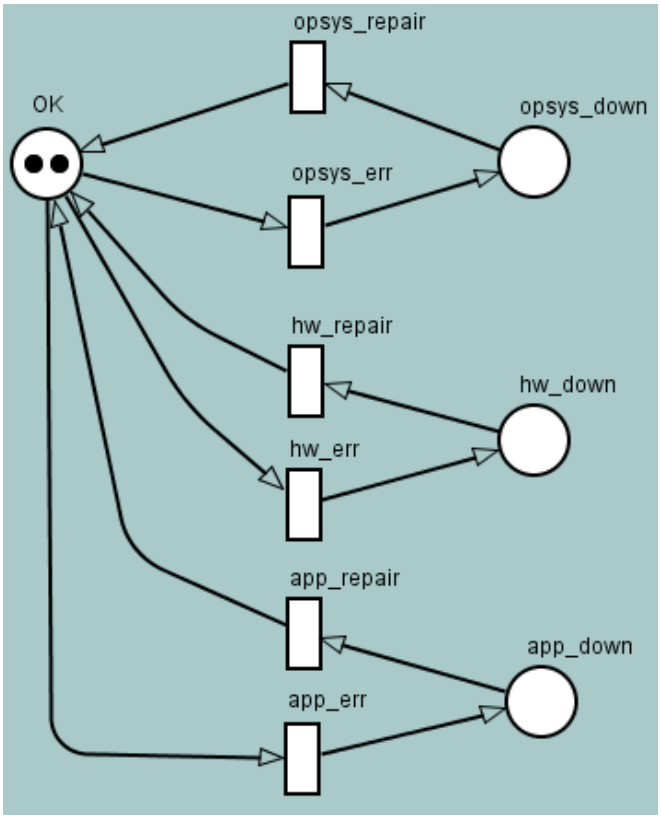
- Nem szabad összekeverni, hogy a fürt viselkedését vagy a fürtöt alkotó egységek viselkedését modellezzük. Maga a fürt és az alkotóegységek más állapotokat járnak be, azonos modellen belül nem szabad összetéveszteni őket. Másrészt modellezési, szemléletbeli döntés, hogy melyiket akarjuk követni a modellel.
- Lényeges elem a fürtszoftver. Már-már filozófiai kérdés, hogy érdemes-e figyelembe venni ennek a hibáit, vagy egyszerű és alaposan letesztelt szoftverelemként szemléljük, amely nem hibásodhat meg? Mindkettő mellett lehetne érvelni, egyrészt a fürtszoftver valóban egyszerű, másrészt akkor jut igazán szerephez, amikor hiba történik a rendszerben – a hiba jellegét nem lehet előre ismerni, tehát elképzelhető, hogy a fürtszoftver nem tudja kezelni. Továbbá, vajon a fürtszoftver egyszeres hibapontnak számít? Ugyan az összes fürttagon külön példányban fut, ám maga a kód azonos, és ezek a példányok mégiscsak egymással szorosan együttműködve egyfajta egységet alkotnak. Érdekes, hogy [44] szerzői a modelljükben figyelembe vették annak a valószínűségét, hogy hiba esetén nem sikerül a fürt újrakonfigurálása; sőt, az érzékenységvizsgálat során arra jutottak, hogy ennek az összetevőnek van a legnagyobb szerepe a rendelkezésre állás fokozásában. Ennek indokát sajnos nem adták meg, ezért csak feltételezhetjük, hogy nem alaptalanul döntöttek így.
- A közös módusú hibák szemszögéből más összetevők is alkothatnak egyszeres hibapontot. Az operációs rendszer például több példányban fut, de feltételezhető, hogy ha ismertté válik valamilyen sebezhetősége, akkor az az összes példányt érinti. Kérdéses, hogy emiatt külön erőforrásként kell kezelnünk az operációs rendszer összes példányát, vagy egyetlen erőforrásként is szemlélhetjük az összest.
- A modellezés nehézsége: nehéz olyan modellt rajzolni, amely a vizsgált rendszer összes lényeges jellemzőjét tükrözi, ugyanakkor áttekinthető és kezelhető. Emellett például az általam használt SPNP eszköz olyan lehetőségeket is támogat, amelyek grafikusan nem ábrázolhatók; márpedig a grafikus modell összeállításának célja éppen a grafikus ábrázolásból fakadó könnyű kezelhetőség biztosítása.
- A 35. ábra szerint a hibák nagyjából fele emberi eredetű. Kérdéses, hogy ezeket miként lehet figyelembe venni a modellezéskor.

6.5.5 Fürt modellje

Egy fürt egyszerűbb modelljét a 46. ábra szerint állíthatjuk fel.

A fenti modellel az operációs rendszer, a hardver és a fürtözött alkalmazás meghibásodásait és javításait tudjuk kezelni. Az OK helyen alapesetben annyi token található, ahány tagja a fürtnek van, jelen esetben kettő. Ha valamelyik összetevő meghibásodik, akkor az egyik fürttag kiesik, és az egyik token valamelyik _down végű névvel ellátott helyre kerül. A javítással a token visszakerül az OK helyre, és a javított

fürttag is üzembe áll. A fürt akkor tudja nyújtani a szolgáltatást, ha az OK helyen legalább egy token található. A modell összes átmenete időzített, exponenciális átmenet.



46. ábra: Egyszerű fürtmodell

A fenti modell több szempontból is korlátozott. Egyrészt nem veszi figyelembe a már említett elemeket, többek közt az emberi és a környezeti hibákat. Másrészt nem tükrözi tökéletesen a rendszer felépítését sem, hiszen ha valamelyik fürttagon szoftverhiba jelentkezik, akkor a modell szerint a javítás elvégzéséig nem hibásodhat meg a fürttag hardvere – a valóságban azonban nem zárható ki az ilyen halmozódás. Ezeket az összefüggéseket tiltó élekkel lehetne ábrázolni, azonban még így is nehéz olyan modellt alkotni, amely az összes egymásra hatást életszerűen tükrözi.

6.5.6 Modellezési paraméterek és eredmények

A modellt az SPNP [46] modellezőeszköz parancssori változatához készítettem el. A modell megjegyzésekkel ellátott kódját és a futtatási parancsokat a függelék tartalmazza. Az átmeneteket a következő táblázatban szereplő paraméterekkel láttam el.

7. táblázat: Modellezési paraméterek

Paraméter neve	Értéke (nap)	Értéke (óra; 1/λ, várható érték)	A λ paraméter értéke
opsys_err	30	720	0,0013889
opsys_repair		0,33	3,030303
hw_err	180	4320	0,0002315
hw_repair		4	0,25
app_err	15	360	0,0027778
app_repair		0,5	2

Bár a fenti értékek eltérnek az 5. táblázat értékeitől, közelítésnek megfelelők, ha figyelembe vesszük az alábbiakat:

- Tekintsünk az operációs rendszer hibájának minden olyan esetet, amikor valóban hiba következik be, illetve amikor hiba nem történik ugyan, de a konfiguráció módosítása vagy frissítés/javítás telepítése miatt újra kell indítani a rendszert.
- Járjunk el hasonlóan az alkalmazás esetében is, feltételezve, hogy a szolgáltatást akár támadások is érhetik.
- Tekintsünk hardverhibának minden olyan hibát, amely a kiszolgálógép hardverét, a hálózati eszközöket vagy a környezetet (helyiség, áramellátás, légkondicionálás, helyiség átrendezése miatti leállás stb.) érinti.

A modellt egy fürttaggal futtatva a készenléti tényező értéke 0,9972, vagyis 99,72% lett; ez 1471 percnyi, kicsivel több, mint egy napi kiesést jelent. Ha kettőre növeltem a fürttagok számát, akkor 0,9999893-as értéket, vagyis erős négykilences, majdnem ötkilences készenléti tényezőt kaptam, ami éves szinten 5,6 percnyi kiesést jelent. Egyértelmű tehát, hogy **fürt építésével jelentős javulást lehet elérni a rendszer rendelkezésre állásában**. Az eredmények összhangban vannak a 4. táblázatban összefoglalt tapasztalati értékekkel is.

Ha a fürttagok számát háromra növeltem, akkor a rendelkezésre állás hétkilencesre nőtt. Ezt az eredményt már fenntartással kell kezelni, hiszen a korábbiakban már láthattuk, hogy a gyakorlatban ilyen rendelkezésre állási szint eléréséhez igen komoly műszaki felkészültség szükséges, és egyetlen apró, akár néhány perces hiba is komolyan visszavetheti a rendelkezésre állást.

6.5.7 Érzékenységvizsgálat

Kézenfekvő kérdés, hogy melyik paraméter javításával tudjuk a legjobb javulást elérni: valamelyik meghibásodási idő kitolásával vagy az egyik javítási idő csökkentésével? Erre érzékenységvizsgálattal kerestem a választ, amelynek során mindegyik paraméter

értékét 20%-kal változtattam meg, és kerestem azt, amelynek a módosítása a legnagyobb változást hozta a rendelkezésre állásban.

8. táblázat: Paraméterek az érzékenységvizsgálathoz

Paraméter neve	Értéke (nap)	Értéke (óra; 1/λ, várható érték)	A λ paraméter értéke
opsys_err	36	864	0,0011574
opsys_repair		0,264	3,7878788
hw_err	216	5184	0,0001929
hw_repair		3,2	0,3125
app_err	18	432	0,0023148
app_repair		0,4	2,5

A futtatásokkal az alábbi eredményre jutottam. Az eredményeket a hetedik tizedesjegyig vettem figyelembe.

9. táblázat: Az érzékenységvizsgálat eredményei

Módosított paraméter	Készenléti tényező	Változás (%)	Kiesési idő (perc)
opsys_err	0,9999898	0,00005	5,36
opsys_repair	0,9999899	0,00006	5,3
hw_err	0,9999904	0,00011	5,04
hw_repair	0,9999906	0,00013	4,94
app_err	0,9999911	0,00018	4,67
app_repair	0,9999915	0,00022	4,46

A fenti eredmények alapján nem állíthatjuk, hogy az eredeti, 5,6 perces értékhez képest jelentős változást sikerült volna elérni a meghibásodások ritkításával és a javítási idők csökkentésével. A legnagyobb változás az alkalmazáshibák javítási idejének csökkentésével érhető el. Sikerült ugyanakkor átugrani az ötkilences és az éves szinten öt perces értéket, ami viszont – ha éppen olyan szerződést kötöttünk – fontos lehet.

Az eredmények alapján általános igazság nem fogalmazható meg – minden esetben egyedileg kell meghatározni, hogy a paraméterek javítása mekkora erőfeszítést kíván, és mekkora hasznot hajt a szolgáltatáskiesési idő csökkentése. A lehetőségek és a ráfordítandó összegek felmérése után kell eldönteni, hogy érdemes-e erőforrásokat fordítani a rendelkezésre állás javítására, például újabb felügyeleti folyamatok kidolgozására és betanulására vagy éppen tartalék cserealkatrész vásárlására.

7 Összefoglalás

Az előző fejezetekben végigjártam a 2. ábra által szemléltetett rendszertervezési folyamatnak a lényegesebb, a tényleges műszaki és üzleti környezet ismerete nélkül is vizsgálható lépéseit. Összefoglaltam, hogy az üzleti szolgáltatások rendelkezésre állásának növelésére szánt megoldásokkal szemben pontosan milyen elvárásokat fogalmazhatunk meg, majd áttekintettem, hogy milyen elméleti és gyakorlati megoldások születtek a problémára, megépítettem néhány egyszerűbb tesztrendszert, végül kísérletet tettem a kapott rendszerek elemzésére, illetve létjogosultságuk bizonyítására.

A diplomaterv-kiírásnak megfelelően, miután áttekintést adtam az olvasónak a rendelkezésre állás és a fürtözés fogalmáról, egyrészt elméleti szempontból megvizsgáltam a fürtözött rendszerek két kisebb részhalmazát, a hálózati terheléselosztási és a feladatátvételi fürtöket, másrészt a Windows Server 2003 operációs rendszerben található szolgáltatásokra támaszkodva megépítettem háromféle tesztrendszert, amelyeken egyszerű méréseket is végeztem. Bár a tesztrendszerek Microsoft termékekre épültek, természetesen nem volt célom egyetlen gyártó szemléletéhez kötődni, ezért röviden más gyártók megoldásait is tárgyaltam. Utaltam arra is, hogy egyrészt a nagy rendelkezésre állású rendszerek építése az adatbázis-kezelés terén különösen nagy hangsúlyt kapott, így további munkát jelenthetne az erre a szűkebb területre kidolgozott számtalan ötlet, megoldás, topológia és eszköz megismerése; másrészt a fürtözési technikák egy mai informatikai infrastruktúrában több helyen, akár több szinten is megjelenhetnek, és az ilyen összetett rendszerek felépítése, felügyelete – éppen összetettségük miatt – alapos felkészülést igényel.

A dolgozatom ötödik és hatodik fejezetében a fürtözési megoldások és általában a rendelkezésre állás témakörének vizsgálatára összpontosítottam, kitérve azokra az értelmezési problémákra is, amelyek a gyakorlatban például a szolgáltatási szerződések megkötésekor felmerülhetnek. Rámutattam, hogy a való életben súlyos problémát jelent a megbízhatósági adatok gyűjtése és értékelése – véleményem szerint olyan terület ez, amelyen a számítógépes rendszerek rendelkezésre állásának javítása érdekében az elkövetkező években komoly fejlesztésekre lesz szükség. Bemutattam, hogy a meglévő és tervezett rendszerek hiányosságai hibafák segítségével egyszerű módon, akár számottevő előzetes felkészülés nélkül is feltárhatók, majd a kiírás negyedik pontját teljesítve, Petri-háló alkalmazásával formális eszközökkel is érvet szolgáltattam amellett, hogy fürtözött rendszerekkel nagyságrendi változást lehet elérni a szolgáltatások rendelkezésre állásában.

Remélem, hogy a dolgozatom alapján az érdeklődő olvasó szert tudott tenni egy olyan átfogó képre erről a szép és változatos témáról, amelynek alapján tovább tud lépni a további megoldások, tervezési minták, bevált konfigurációk megismerése és gyakorlati alkalmazása felé.

8 Források

[1] Dr. Majzik István: Informatikai rendszerek szolgáltatásbiztonsága tantárgy, oktatási jegyzet, VIMM4324, Budapesti Műszaki és Gazdaságtudományi Egyetem, 2006

[2] Wikipedia, „Computer cluster”, http://en.wikipedia.org/wiki/Computer_cluster

[3] Microsoft Corporation: Microsoft Solution for Internet Business, Performance and Capacity Planning, Part 3 – MSIB 2.0 Site Availability, 2006. URL: http://www.microsoft.com/technet/solutionaccelerators/citsrv/ib/msib2tca.mspx

[4] Dominique A. Heger: Reliability Engineering - Business Implication, Concepts, and Tools; URL: http://fortuitous.com/docs/primers/RelEng_Primer.pdf

[5] Microsoft Corporation: Planning for Reliability and High Availability, URL: http://msdn2.microsoft.com/en-us/library/ms942932.aspx

[6] Tim Read, Gia-Khanh Nguyen, Bob Bart: Sun™ Cluster 3.2 Software: Making Oracle Database 10g R2 RAC Even More „Unbreakable”, URL: http://www.sun.com/software/whitepapers/solaris10/solaris_cluster.pdf

[7] Bradley Mitchell, Cluster Network Computing Architecture, URL: http://compnetworking.about.com/od/networkdesign/l/aa041600a.htm

[8] Server Clusters : Architecture Overview for Windows Server 2003, URL: http://download.microsoft.com/download/0/a/4/0a4db63c-0488-46e3-8add-28a3c0648855/ServerClustersArchitecture.doc

[9] Microsoft Corporation, Microsoft Windows Server TechCenter, Server Load, URL: http://technet2.microsoft.com/WindowsServer/en/library/4361a756-bd0a-41ff-8820-c6f83ad36c171033.mspx?mfr=true

[10] Oracle Corporation: Oracle® Database High Availability Architecture and Best Practices, elméleti és gyakorlati jellegű útmutatás, URL: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10726/overview.htm

[11] Hewlett-Packard Development Company, Three Data Center Architectures, URL: http://docs.hp.com/en/B7660-90014/ch02s03.html

[12] Sun Microsystems, Inc., Sun Cluster 3.2 Documentation Center, URL: http://docs.sun.com/app/docs/doc/820-0335/6nc35dge4?a=view

[13] Sun Microsystems, Inc., Campus Clustering With Sun Cluster Software, URL: http://docs.sun.com/app/docs/doc/819-2993/6n58bhel7

[14] Sun Microsystems, Inc., Sun Cluster Geographic Edition, Documentation Center, URL: http://docs.sun.com/app/docs/doc/820-0617/6ncbibf24?a=view

[15] Oracle Corporation, Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide, 10g Release 2, URL: http://download-uk.oracle.com/docs/cd/B19306_01/rac.102/b14197/toc.htm

[16] Microsoft Solution for Internet Business, Performance and Capacity Planning, URL: http://www.microsoft.com/technet/solutionaccelerators/citsrv/ib/msib2tca.mspx

[17] Microsoft Corporation: SQL Server 2000 High Availability Series, Introduction, URL: http://www.microsoft.com/technet/prodtechnol/sql/2000/deploy/harag01.mspx

[18] Cristina Simache, Mohamed Kaâniche, and Ayda Saidane: Event Log based Dependability Analysis of Windows NT and 2K Systems

[19] Fujitsu Siemens Computers, Five Nines, tanulmány, 2002. decemberi kiadás

[20] Jun Xu, Zbigniew Kalbarczyk, Ravishankar K. Iyer: Networked Windows NT System Field Failure Data Analysis

[21] Sun Microsystems, Availability Modeling for the Sun™ Java System Application Server Enterprise Edition 7, technikai tanulmány, URL: http://www.sun.com/software/products/appsrvr/AS7EEHA0504.pdf

[22] Microsoft Corporation, Súgó és támogatás, An update is available that adds a file share witness feature and a configurable cluster heartbeats feature to Windows Server 2003 Service Pack 1-based server clusters, URL: http://support.microsoft.com/kb/921181

[23] Hewlett-Packard Development Company, HP NonStop server virtualization primer

[24] Gordon Haff, Itanium goes nonstop at HP, Quick note, 2005. június 13.

[25] Hewlett-Packard Development Company, Clarifying HP NonStop server new terminology

[26] Round-robin DNS megvalósítása a BIND névkiszolgáló szoftverrel, URL: http://content.websitegear.com/article/load_balance_dns.htm

[27] Luis Aversa és Azer Bestavros: Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting, Technical Project Report, URL: http://www.cs.bu.edu/~best/res/projects/DPRClusterLoadBalancing/

[28] Yiu-Fai Sit, Cho-Li Wang és Francis Lau: Socket Cloning for Cluster-Based Web Servers, URL: http://www.cs.hku.hk/~clwang/papers/cluster2002-FNL-socket-cloning.pdf

[29] Microsoft TechNet, Network Load Balancing Technical Overview http://www.microsoft.com/technet/prodtechnol/acs/reskit/acrkappb.mspx

[30] The High Availability Linux Project, URL: http://www.linux-ha.org

[31] Brien M. Posey: Understanding How Cluster Quorums Work, URL: http://www.windowsnetworking.com/articles_tutorials/Cluster-Quorums.html

[32] VMware tudásbázis, Microsoft NLB Not Working Properly in Unicast Mode, URL: http://kb.vmware.com/selfservice/viewContent.do?externalId=1556

[33] Cisco Systems, Inc., dokumentáció az IP protokoll csoportcímzési funkcióiról, URL: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ipmulti.htm

[34] Database Journal, Tarry Singh: Oracle RAC Administration – Part 16: Balancing act between Server and Client, gyakorlati útmutató rendszergazdáknak, URL: http://www.databasejournal.com/features/oracle/article.php/3663401

[35] Oracle Real Application Clusters 10g Release 2, Technical Comparison with Microsoft SQL Server 2005, technikai termék-összehasonlító cikk, URL:

http://www.oracle.com/technology/products/database/clustering/pdf/twp_rac_compare_sqlserver2005.pdf

[36] Oracle Corporation, Oracle Real Application Clusters on Extended Distance Clusters, 2006. októberi Oracle tanulmány a 10gR2 verzióhoz, URL: http://www.oracle.com/technology/products/database/clustering/pdf/ExtendedRAC10gR2.pdf

[37] Hewlett-Packard Development Company, HP Integrity NonStop servers, Family guide, összefoglaló a NonStop kiszolgálói termékcsalád tagjairól URL: http://h71028.www7.hp.com/ERC/downloads/4AA0-5422ENW.pdf

[38] Failover Clustering with Windows Server "Longhorn", URL: http://www.microsoft.com/windowsserver/longhorn/failover-clusters.mspx

[39] A Packetyzer alkalmazás (ingyenes) letöltése a Network Chemistry cégtől, URL: http://www.networkchemistry.com/products/packetyzer.php

[40] A PanTel Kft szolgáltatás minőség mutatói 2005 II. félévben, URL: http://www.pantel.hu/fileadmin/doc_docs/PanTe_szolg%E1ltat%E1s_min%F5s%E9g_mutat%F3i-2005.doc

[41] Előre konfigurált, szabadon letölthető, linuxos virtuális gép VMware alá, iSCSI-céllal, URL: http://www.vmware.com/vmtn/appliances/directory/217

[42] A [41] virtuális gépben is használt iSCSI-cél fejlesztési projektjének weblapja, URL: http://iscsitarget.sourceforge.net/

[43] Solt György: Valószínűségszámítás; Műszaki Könyvkiadó, Budapest, Bolyai-könyvek sorozat

[44] Sun Microsystems, Modeling Sun Cluster Availability, tanulmány, URL: http://www.phptr.com/articles/article.asp?p=31756&seqNum=1&rl=1

[45] A TimeNet modellező eszköz honlapja, Berlini Műszaki Egyetem, URL: http://pdv.cs.tu-berlin.de/~timenet/

[46] Az SPNP modellező eszköz honlapja, Dr. Kishor S. Trivedi, Duke Egyetem, URL: http://www.ee.duke.edu/~kst/

Függelék

Az SPNP parancssori változatánál a Petri-hálókat, a vizsgálati paramétereket, a rewardokat stb. a programcsomagban található API-k felhasználásával, C stílusú fájlban kódolhatjuk. A 46. ábra: Egyszerű fürtmodell modelljét az alábbi kóddal valósítottam meg.

1. kódrészlet: A modellezésre használt modell.c fájl kódja

```
# include "user.h"
/* Globális változók */

int clumembers; //fürttagok száma
double opsys_err; //az opsys_err átmenet paramétere
double opsys_repair; //az opsys_repair átmenet paramétere
double hw_err; //a hw_err átmenet paramétere
double hw_repair; //a hw_repair átmenet paramétere
double app_err; //az app_err átmenet paramétere
double app_repair; //az app_repair átmenet paramétere

void options() {
/* Beállítások megadása, lényegében alapbeállítások */
       iopt(IOP_SSMETHOD,VAL_SSSOR);
       iopt(IOP_PR_FULL_MARK,VAL_YES);
       iopt(IOP_PR_MARK_ORDER,VAL_CANONIC);
       iopt(IOP_PR_MC_ORDER,VAL_TOFROM);
       iopt(IOP_PR_MC,VAL_YES);
       iopt(IOP_MC,VAL_CTMC);
       iopt(IOP_PR_PROB,VAL_YES);
       iopt(IOP_PR_RSET,VAL_YES);
       iopt(IOP_PR_RGRAPH,VAL_YES);
       iopt(IOP_ITERATIONS,2000);
       iopt(IOP_CUMULATIVE,VAL_NO);
       fopt(FOP_ABS_RET_M0,0.0);
       fopt(FOP_PRECISION,0.00000001);
/* Bemeneti értékek beolvasása a parancssorból */
       clumembers = input("A fürttagok száma:");
       opsys_err = finput("opsys_err értéke:");
       opsys_repair = finput("opsys_repair értéke:");
       hw_err = finput("hw_err értéke:");
       hw_repair = finput("hw_repair értéke:");
       app_err = finput("app_err értéke:");
       app_repair = finput("app_repair értéke:");
}
```

```
void net() {
/* Helyek megadása */
       place("OK");
       place("opsys_down");
       place("hw_down");
       place("app_down");
/* Átmenetek */
       trans("opsys_err");
       trans("opsys_repair");
       trans("hw_err");
       trans("hw_repair");
       trans("app_err");
       trans("app_repair");
/* Kezdeti tokenszám megadása */
       init("OK",clumembers);
/* Átmenetek paraméterei */
       ratedep("opsys_err",opsys_err,"OK");
       rateval("opsys_repair",opsys_repair);
       ratedep("hw_err",hw_err,"OK");
       rateval("hw_repair",hw_repair);
       ratedep("app_err",app_err,"OK");
       rateval("app_repair",app_repair);
/* Élek: iarc(átmenetbe helyből), oarc(átmenetből helyre) */
       iarc("opsys_err","OK");
       oarc("opsys_err","opsys_down");
       iarc("opsys_repair","opsys_down");
       oarc("opsys_repair","OK");
       iarc("hw_err","OK");
       oarc("hw_err","hw_down");
       iarc("hw_repair","hw_down");
       oarc("hw_repair","OK");
       iarc("app_err","OK");
       oarc("app_err","app_down");
       iarc("app_repair","app_down");
       oarc("app_repair","OK");
}

int assert() {
       return RES_NOERR;
}

void ac_init() {
}

void ac_reach() {
```

```
}
/*Rewardfüggvény*/
double rd() {
	return ((mark("OK")>0) ? 1:0);
}
/* Itt történik a megoldás kiszámítása */
void ac_final() {
	solve(INFINITY);
	pr_expected("Készenléti tényezo: ", rd);
}
```

A kódból két elem igényelhet további magyarázatot:

- ratedep: Olyan átmenet megadására használható, amelynek a számításkor alkalmazott paramétere függ attól is, hogy a megadott helyen hány token található. A meghibásodást jelentő átmeneteknél ilyet kellett használni, hiszen két vagy három számítógép közül nagyobb valószínűséggel hibásodik meg egy, mintha csak egyetlen gépnek a hibáit figyelnénk.
- rateval: Olyat átmenet megadására alkalmas, amelynél a paramétert önmagában alkalmazzuk. A javításoknál ilyet használtam, hiszen a javítás intenzitása nem függ a javításra váró egységek számától.

A modell futtatására a következő tartalmú batch fájlt használtam (spnp.bat):

```
make -f %SPNP_DIRECTORY%\Makerun.dos SPN=%1
```

Az érzékenységvizsgálat egyszerűbbé tétele érdekében a bemenő paramétereket is fájlokban adtam meg. Ezek egyszerű szövegfájlok voltak, mindegyik soruk egy-egy bemenő paramétert tartalmazott. Például (sens_3.txt, az érzékenységvizsgálat 3. lépéséhez):

```
2
0.0013889
3.030303
0.0001929
0.25
0.0027778
2
```

Bemeneti paraméterfájl használatakor tehát a futtatási parancs a következő volt:

```
spnp.bat < sens_3.txt
```

Az SPNP a futtatás során kapott eredményeket egy .out kiterjesztésű fájlba írja, ennek neve egyezik a modellfájléval, jelen esetben modell.out volt. A tartalma a következő formátumot követte:

```
INPUT: A fürttagok száma: = 2


INPUT: opsys_err értéke: = 0.0013889


INPUT: opsys_repair értéke: = 3.0303


INPUT: hw_err értéke: = 0.0002315


INPUT: hw_repair értéke: = 0.25


INPUT: app_err értéke: = 0.0027778


INPUT: app_repair értéke: = 2
========================================================================


                          TIME :   INFINITY


========================================================================
EXPECTED: Készenléti tényező:  = 0.999989371628
```