Szolgáltatásbiztonságra tervezés laboratórium (VIMIM236)

Megbízhatósági modellezés

Mérési segédlet

Készítette: Vörös András

Utolsó módosítás: 2010-10-10

Verzió: 1.0

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

1 Bevezető

A megbízhatósági modellezés egyre fontosabb szerepet kap napjaink informatikai infrastruktúráinak tervezésében. Ahogy a szolgáltatásokra egyre nagyobb feladat hárul, kiesésük is egyre költségesebb a cégeknek. Az alábbi táblázatban egy tanulmány adatai szerepelnek 2003-ból, melyben jól látható, hogy már évekkel ezelőtti is komoly költségekkel járt egyes iparágakban a szolgáltatások kiesése:

Esettanulmány	**Éves bevétel**	**Kiesés költsége**	**Költség/óra**
Energiaszektor	6.75 milliárd $	4.3 millió $	1624 $
„High tech”	1.3 milliárd $	10.2 millió $	4,167 $
Egészségügy	44 milliárd $	74.6 millió $	96,632 $
Utazás	850 millió $	2.4 millió $	38,710 $
Pénzügyi szektor (USA)	4.0 milliárd $	10.6 millió $	28,342 $
Pénzügyi szektor (Európa)	1.2 milliárd $	379,000 $	1573 $

A következő táblázat frissebb adatokat tartalmaz, egy 2009-es felmérés iparágak és tevékenységek szerint lebontva vizsgálta az üzleti tevékenységek veszteségeit a rendszerek megbízhatatlansága miatt:

Iparág	**üzleti tevékenység**	**átlagos óránkénti kiesés költsége**
pénzügyi szektor	ügynöki, közvetítői	6.45 millió $
pénzügyi szektor	hitelkártya, pénzügyi tranzakciók	2.6 millió $
média	pay-per-view	150 000 $
kiskereskedelem	TV shopping	113 000 $
utazás	repülőjegy foglalás	90 000 $

Hogy ezek a költségek tervezhetőek és előre jósolhatóak legyenek, szükségünk van a megbízhatósági modellezésre, és a mögöttes matematikai apparátusra.

A mérési segédlet célja, hogy megalapozza a mérést a modellezési alapismeretek felfrissítésével, továbbá hogy a mérés során segítséget nyújtson az eszközök használatához. A mérés épít a korábban a tárgyhoz tartozó előadások során elsajátított ismeretekre.

2 Modellezési formalizmusok

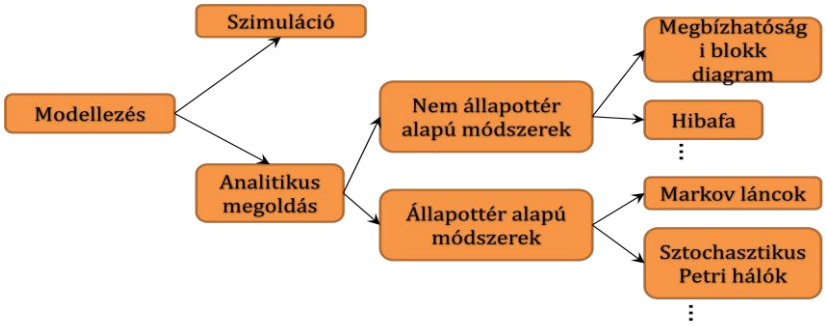
A megbízhatóság számítására többféle lehetőségünk van:

- szimuláció
- analitikus megoldás

A szimuláció előnye, hogy bármilyen modellre alkalmazható, nincsenek olyan szoros megkötések (eloszlások, modellezhető viselkedések), mint ami az analitikus megoldhatóságot jellemzi. Nagy hátránya azonban, hogy csak korlátozott az így nyert információ, sok esetben nem eldönthető, hogy elég esetre, időre futtattuk-e a szimulációt. Az így kapott eredményeket óvatosan kell kezelni.

Az analitikus megoldás előnye, hogy pontos eredményt ad, ellenben sok modellre nem alkalmazható, különösen a dinamikus modelleknél vannak korlátai.

A különböző lehetőségeket a következő ábra szemlélteti:



1. ábra: Megbízhatósági modellek

A továbbiakban végignézzük a mérés során a megbízhatósági modellezéshez használt eszközöket, példák segítségével megvizsgáljuk a modellezési és analízis lehetőségeket. A mérés során hasonló feladatokat kell majd megoldani.

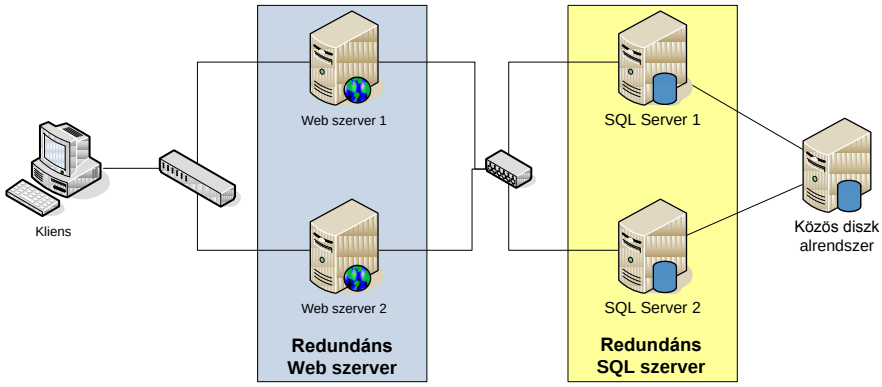
3 Nem állapottér alapú modellek

Az alábbi fejezetben a **SHARPE** eszköz segítségével áttekintjük a hibafa-rajzolást, és a hibafák által nyújtott analízis lehetőségeket, majd áttérünk a másik gyakran használt nem állapottér alapú analízisre, a megbízhatósági blokk diagramokra (RBD), és megnézzük a mintapélda RBD modelljét.

3.1 Infrastruktúra

A következőkben egy egyszerű számítógépes infrastruktúra hibafáját nézzük meg. A példa infrastruktúra egy egyszerű webes és egyéb szolgáltatásokat nyújtó hálózat leegyszerűsített modellje.

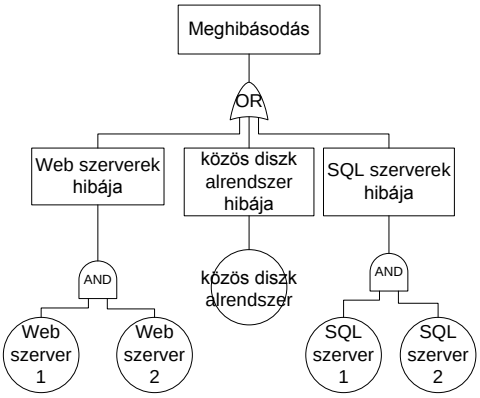
Az infrastruktúra egy webszerver fürtből, egy SQL szerver fürtből és az adatbázis szerverek által használt közös diszk alrendszerből áll. Ez a webszerverek és az SQL szerverek esetén is redundáns megoldás növeli a megbízhatóságot, hiszen ha kiesik az egyik szerver, attól még a teljes szolgáltatás nem esik ki. A példa során megnézzük, hogy mi vezethet az infrastruktúra leállásához.



2. ábra: webes infrastruktúra

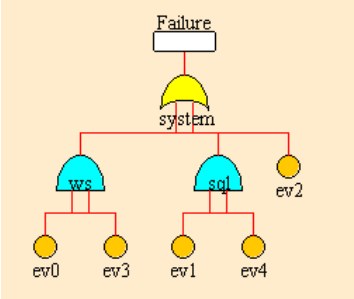
3.2 Hibafa rajzolás

A feladat során először felvesszük a legfelsőbb szintű eseményt: ez lesz a teljes rendszerhiba. A rendszer működéséhez legalább egy web szervernek, egy SQL szervernek és a közös diszk alrendszernek működnie kell.



3. ábra: webes infrastruktúra hibafája

A fenti hibafa **SHARPE** eszközben megrajzolt változata (*ev* események a komponensek meghibásodásai):



4. ábra: webes rendszer SHARPE hibafája

Az ábrán látható, hogy a SPOF (Single Point of Failure, egyszeres meghibásodási pont) a közös diszk alrendszer, hiszen a meghibásodása már önmagában elég a rendszer leállásához.

Megjegyzések a **SHARPE** eszköz használatához:

- Egy kaput, ha már letettünk, akkor csak törölni tudjuk, sem a nevét, sem a bemenő lábainak számát nem tudjuk változtatni (a legfelső szintű kapuknál érdemes előre eltervezni, hogy hány lábra is lesz szükség)
- A kapuk neve sem „*and*”, sem „*or*” (sem „*AND*” és „*OR*”) nem lehet, és ha már adtunk nevet a kapuknak, a nevüket megváltoztatni nem tudjuk, tehát érdemes nem rögtön a legelején elrontani egy rossz névvel
- Az elemek neveiben sem szóköz, sem kötőjel, sem ékezet vagy bármilyen speciális karakter nem szerepelhet
- Mielőtt átlépünk egy másik eseményre a szerkesztő ablakban, nyomjuk meg a *Validate* gombot! Az értékeket akkor fogadja el, ha narancssárga lesz az esemény.
- Tizedespontot kell használni és nem tizedesvesszőt.
- Új kapu az eseményre vagy csomópontra való kattintással adható hozzá.

3.3 Megbízhatósági mérés:

A komponensek megbízhatóságát λ paraméterű exponenciális eloszlással szoktuk jellemezni, ahol a meghibásodási idő várható értéke 1/ λ.

Rendelkezzenek a komponensek a következő megbízhatósági adatokkal:

komponens	λ
webszerver	0.05
SQL szerver	0.01
közös diszk	0.2

Ebből a **SHARPE** segítségével kiszámolhatjuk a megbízhatósági görbét:

Az *Analysis Editor →Analysis* menüpontját kiválasztva jutunk el analízis ablakhoz.

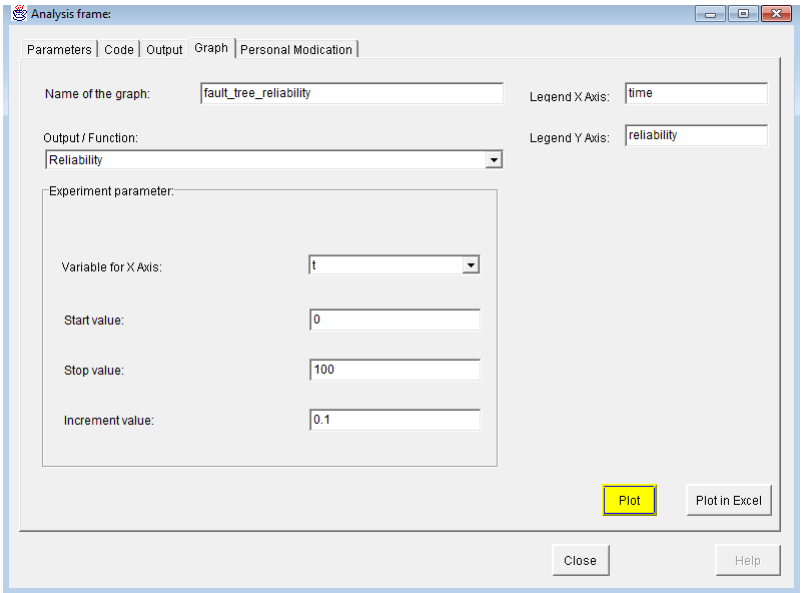
Ha az analízis ablakban a *Parameters* fülre kattintunk, akkor a következő értékeket számolhatjuk ki a programmal:

- *Reliability*, azaz megbízhatóság (egy adott időpillanat)
- *Unrelability*
- *Main Time to Failure* (MTTF), azaz a meghibásodás várható ideje
- *Variance*, azaz szórásnégyzet

Ha az előbb látott modell esetén megkérdezzük a programtól, hogy mi a megbízhatóság várható értéke a $t = 10$ időpillanatban, akkor a következő kimenetet kapjuk:

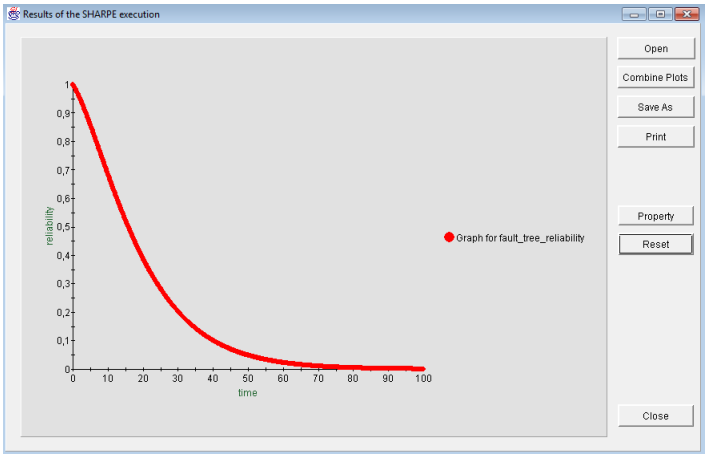
```
**********************************************************************
 *********  Outputs asked for the model: ft **************
 Reliability at time 10
 Reliability:   1.13347087e-001
```

Ha az analízis ablakban a *Graph* fülre kattintunk, akkor a következő ablakhoz jutunk:



5. ábra: Hibafa analízis beállítások

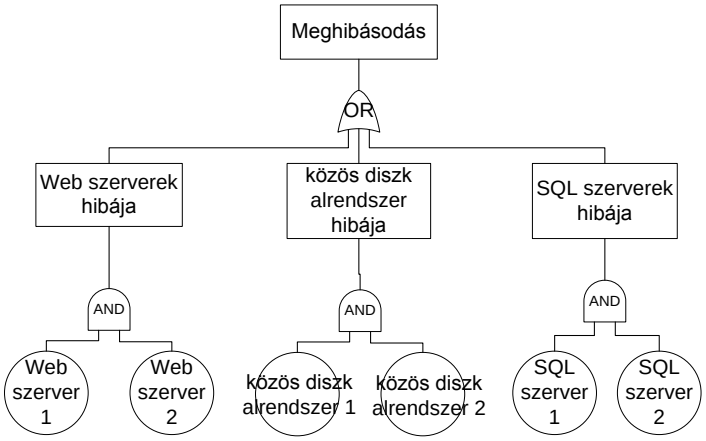
Fontos a paraméterek megfelelő kitöltése, nem érdemes sokkal tovább futtatni a megjelenítést, mint ahol még releváns eredményeket láthatunk. A paramétereket megfelelően beállítva, majd a *Plot* gombra kattintva a következő ábrát kapjuk.



6. ábra: Hibafa megbízhatósága

Az így kapott diagramot elmenthetjük a program segítségével, ha a *Save As* gombra kattintunk. Ez akkor praktikus, ha később majd több függvényt szeretnénk egy diagramon megjeleníteni.

Vizsgáljuk meg, hogy ha a közös diszk alrendszert is redundánssá tesszük, akkor hogyan változik meg a rendszer megbízhatósága. Ehhez a diszk alrendszernek létrehozunk egy AND kaput, mellyel összekötjük a különálló diszk alrendszer eseményeket:

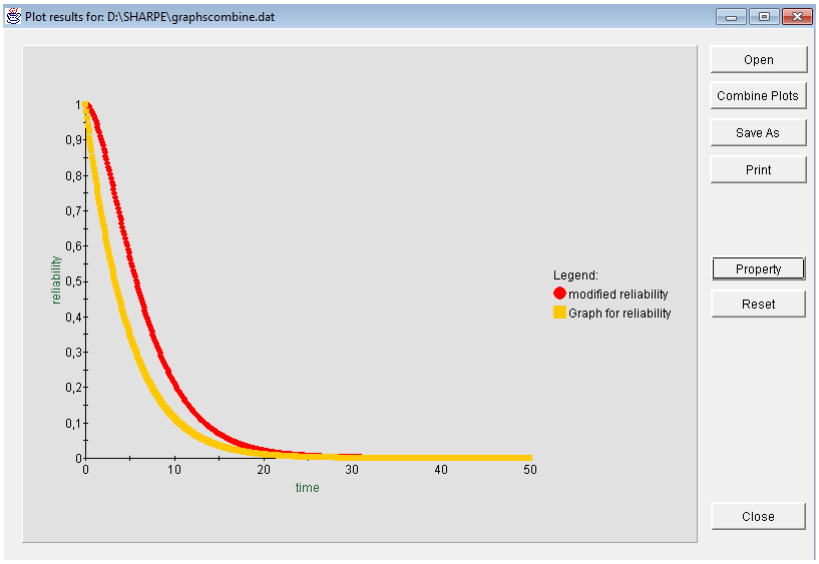


7. ábra: Módosított webes infrastruktúra hibafája

Ekkor a korábban is említett *Reliability* menüpont a következő kimenetet produkálja:

```
**********************************************************************
 *********  Outputs asked for the model: ft **************
 Reliability at time 10
 Reliability:   2.11354313e-001
----------------------------------------
```

Ha meg szeretnénk jeleníteni a korábbi és a módosított megbízhatósági értékeket egy ábrán, akkor először megjelenítjük a módosított modell megbízhatósági függvényét a megfelelő paraméterek beállítása után a *Plot* gombra kattintva. Fontos, hogy a két ábra felbontásának egyeznie kell, tehát ugyanaddig a pontig kell futtatni a megjelenítést, különben a **SHARPE** hibát fog jelezni. Ha megjelenítettük a módosított modell megbízhatósági függvényét, akkor a *Combine Plots* gombra kattintva megnyithatjuk mellé a korábban elmentett régi megbízhatósági függvényünket, amit a program utána egy közös ablakban megjelenít:



8. ábra: Modellek összevetése megbízhatóság szempontjából

Az ábrán sárgával jelöltük a rendszer megbízhatóságát, és pirossal a módosított modell alapján számolt megbízhatósági függvényt.

Jól látható, hogy növekedett a rendszer megbízhatósága.

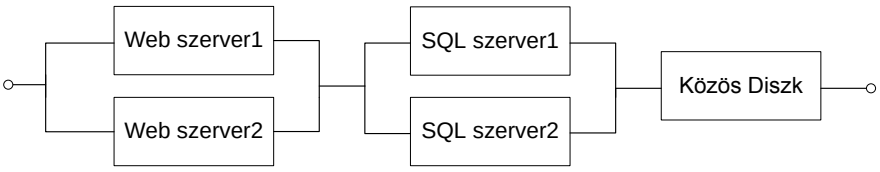
3.4 Megbízhatósági Blokk Diagram (Reliability Block Diagram, RBD)

A **SHARPE** eszközben elkészíthetjük a rendszer megbízhatósági blokk diagramját, azaz RBD-jét. Ehhez érdemes új *project*-et létrehozni, de a **SHARPE** lehetőséget nyújt akár arra is, hogy hierarchikusan kombináljuk a különböző modellezési formalizmusokat, azaz RBD-vel leírhatjuk az egyik komponensen (hibafa komponens) bekövetkező eseményt. Mi most a példában egy külön project-et hozunk létre.

Az RBD-k sorosan és párhuzamosan kötött modellelemekből állnak. Az RBD-kben azzal a feltételezéssel élünk, hogy meghibásodni csak komponens tud, összeköttetés nem, azaz a modellelemek közötti összeköttetések csak a hibaterjedés logikai leírására szolgálnak. Továbbá a komponensek meghibásodása független.

Ha n darab sorosan kötött elem megbízhatóságai $R_1 \ldots R_n$, akkor az eredő közös megbízhatóság: $R_e = \prod R_n$, ha párhuzamosan kötjük őket, akkor $1 - R_e = \prod(1 - R_n)$.

Ezek alapján a fenti példarendszer RBD-je a következőképpen néz ki:



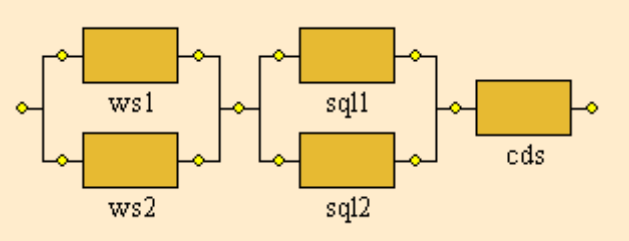
9. ábra: Webes rendszer RBD-je

A modellek SHARPE eszközben történő elkészítésekor:

- a *Series* gombra kattintva, utána pedig az egyik modellelemre kattintva tudjuk az adott modellelemet soros komponensekre bontani
- a *Parallel* gombra kattintva, utána pedig az egyik modellelemre kattintva tudjuk az adott modellelemet párhuzamos komponensekre bontani
- mindkét esetben meg kell adni, hogy hány komponens jöjjön létre

- a *Parallel* esetben megadhatjuk az „*Identical to the current block*” checkbox-ot bejelölve, hogy az új elemek ugyanolyan típusúak legyenek, mint a kiinduló elem. Esetünkben, mivel mindkét webszerver és sql szerver ugyanolyan típusú, alkalmazhatjuk.
- kiinduló állapotban egy blokk van, és ezt bontjuk tovább, ezért figyelni kell, hogy ha a modellünk alapvetően soros viselkedésű, akkor először soros modellelemekre bontsuk, ha párhuzamos viselkedésű, akkor párhuzamos elemekre

A **SHARPE** eszközben elkészített modell a következőképpen fog kinézni:



10. ábra:Webes rendszer SHARPE RBD-je

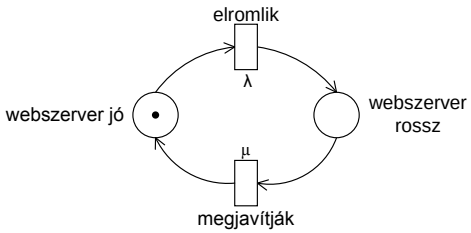
Ha erre a modellre lefuttatjuk az analíziseket a fentebb (hibafáknál) megadott paraméterekkel, akkor ugyanazokat az eredményeket fogjuk kapni.

4 Petri háló modellezés, mintapélda

A következőkben a fentebb említett webes architektúra egy részének Petri háló modelljét fogjuk megvizsgálni. Hogy a Petri hálók tulajdonságait jobban kihasználjuk, nem megbízhatóságot, hanem rendelkezésre állást fogunk számolni. Ehhez újabb paraméterre lesz szükségünk, nem csak a meghibásodási valószínűséget, hanem a javítási idő várható értékét is meg kell adnunk. A táblázatban szereplő μ ennek az exponenciális eloszlású javítási időnek a paramétere, ahol az eloszlás várható értéke 1/μ.

komponens/intenzitás	λ	μ
webszerver	0.05	0,5

Először tekintsük egy webszerver Petri háló modelljét. Alapvetően két állapota van: egy jó és elromlott állapot, amit két hellyel jelölünk („*webszerver jó”* és „*webszerver rossz”*), melyek között az „*elromlik”* és a „*megjavítják”* tranzíció viszi át a tokeneket.



11. ábra: Egyszerű webszerver Petri hálója

Nyilván ha egy webszerverből áll csak a rendszer, akkor könnyen meg tudjuk határozni a rendszer állapotát: ha jó a webszerver, akkor működő állapotban vagyunk, egyébként pedig nem. Továbbá az állandósult állapotban az állapotok valószínűségét is könnyen meghatározzuk:

Állandósult állapotban a következő jól ismert egyenletet kell megoldani: $p(t) = p(0) \cdot e^{-Qt}$ (valószínűség időfüggvénye exponenciális eloszlású átmenetek esetén). Tehát $e^{-Qt} = 1$ (ahol Q az állapot-átmeneti intenzitásmátrix), azaz a Q mátrix 0 sajátértékhez tartozó sajátvektorát keressük:

$$(a, b) \cdot Q = (0,0)$$

Jelen esetben: $Q = \begin{matrix} -\lambda & \lambda \\ \mu & -\mu \end{matrix}$

Az egyenlet megoldásai: $a = \frac{\mu}{\lambda+\mu}$, $b = \frac{\lambda}{\lambda+\mu}$

Ebbe behelyettesítve megkapjuk a webszerver rendelkezésre állását, azaz az idő mekkora részében lesz elérhető a webszerver.

4.1 A modell megalkotása TimenNET-ben

Létrehozunk egy új *eDSPN* modellt.

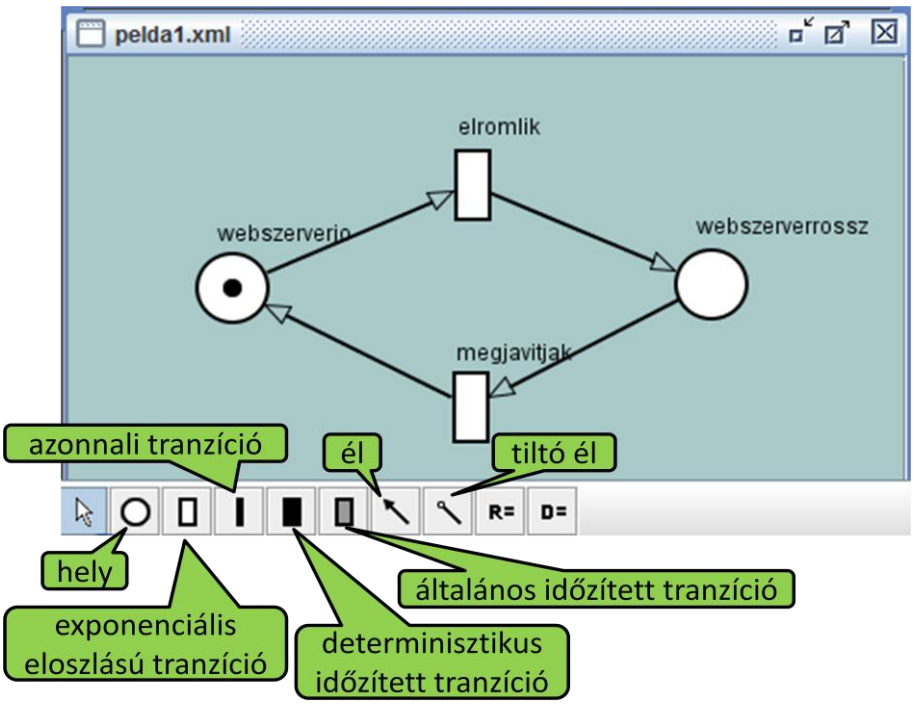
eDSPN:

- „Extended Deterministic and Stochastic Petri Net” – kiterjesztett determinisztikus és sztochasztikus Petri háló
- exponenciális eloszlású eseményekre (tranzíció) nincs megkötés, hiszen a Markovi folyamatok jól vizsgálhatóak
- minden egyes állapotban (marking) egy nem exponenciális eloszlású (expolinomiális) esemény (tranzíció) lehet maximum engedélyezett, azaz a nem exponenciális eloszlású események kölcsönösen ki kell, hogy zárják egymást
- a nem exponenciális eloszlású tranzíciók lehetnek determinisztikusak, egyenletes eloszlásúak, háromszög eloszlásúak, stb…
- az azonnali (immediate) tüzelések, ha engedélyezettek, akkor az időzített tranzíciók előtt, azonnal végrehajtódnak. Ha több azonnali tüzelés is engedélyezett, akkor prioritást kell definiálni közöttük

Az eszköz külön tranzíciós jelölést használ

- az exponenciális eloszlású,
- az azonnali,
- a determinisztikus időzített és az
- általános sztochasztikus időzített tranzíciókra,

amely a következő, a webszerver **TimeNet** eDSPN modelljét ábrázoló képen látszik.



12. ábra: TimeNET felhasználói felület

A tranzíciók tüzelési valószínűségét a tranzícióra kattintva tehetjük meg, ekkor a **TimeNET**-ben a jobb felső sarokban a következő beállítási lehetőségek jelennek meg:

Qualified Name	Value
text	elromlik
delay	20
serverType	ExclusiveServer
preemptionPolicy	PRD
DTSPNpriority	1

13. ábra: Tranzíció beállítások

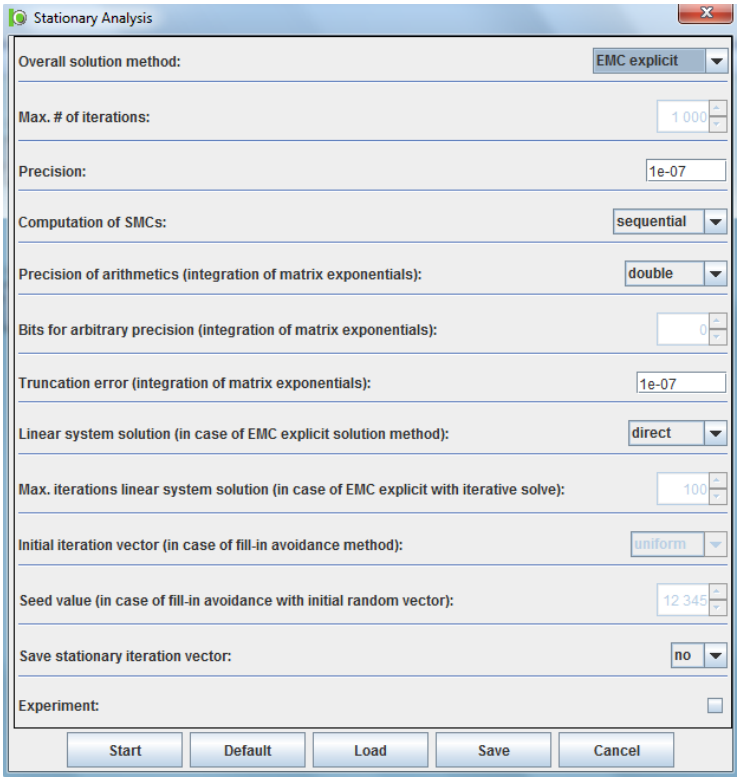
- *text*: a tranzíció neve, ami megjelenik a Petri hálóban a kezelői felületen
- *delay*: a tranzíció eloszlásának várható értéke, ami exponenciális esetben egyenlő $1/\lambda$-val
- *serverType*: a tranzíciók kiértékelésének módja, mi itt a példában szekvenciális kiértékelésű „*ExclusiveServer*" beállítást használjuk, amikor is az engedélyezett tranzíciók nem konkurensen értékelődnek ki. „*InfiniteServer*" beállítást akkor használunk, ha több token van egy helyen, és ezek konkurrensen tüzelhetőek.
- *preemptionPolicy*: azt határozza meg, ha egy tranzíció az engedélyezett volt, megszakították, majd újra engedélyezett lesz, hogyan viselkedjen. *PRD* (preemptive repeat different) módban a korábban eltelt idő elveszik, és a tüzelési periódus újrakezdődik, míg *PRS* (preemptive resume) módban a tüzelési idő onnan folytatódik, ahol egyszer már megszakították
- *DTSPNpriority*: azoknál az analízis módszereknél, ahol a Petri háló analízise során diszkrét idő – modellt használunk, előfordulhat, hogy egyszerre több tranzíció konkurensen engedélyezett lesz – ilyenkor a háló további működése szempontjából ezt a konfliktust fel kell oldani, melyet priotitásokkal tudunk megoldani. Ilyenkor az exponenciális eloszlást a diszkrét megfelelőjével, a geometriai eloszlással közelítünk. Esetünkben azonban nincs erre szükség.

Különböző analíziseket végezhetünk az elkészített modellen. A *Validate* menü *Check Structure* pontját kiválasztva ellenőrizhetjük, hogy a modell milyen strukturális tulajdonságokkal rendelkezik,

kiszámíthatjuk a P invariánsokat, továbbá a konfliktusban lévő azonnali tüzelésű (immediate) tranzíciókat is megkaphatjuk. Fontos, hogy az ilyenkor felugró ablakot ne a felső X-re kattintva zárjuk be, mert akkor a **TimeNET** is bezáródik, hanem az ablakban lévő *Close* gombot használjuk.

- A *Validate* menüben az *Estimate Statespace* opciót választva a program bejárja a Petri háló állapotterét, és egy becslést ad a méretére. Fontos: ha nagy az állapottér, akkor bonyolultabb az analízis, tehát várhatóan tovább is fog futni.
- Sztochasztikus analízisekre az *Evaluation* menüben van lehetőségünk: állandósult állapotbeli és tranziens analíziseket is végezhetünk.

Esetünkben, ha az *Evaluation* menü *Stationary Analysis* menüpontjára kattintunk, akkor a következő ablak nyílik meg:



14. ábra: Állandósult állapotbeli analízis beállítások

Itt beállíthatjuk az állandósult állapotbeli (*Steady State, Stationary*) analízis jellemzőit.

A legfontosabb beállításaink:

- *Overall solution method*: EMC explicit, mely azt mondja meg, hogy a beágyazott Markov láncot hogyan kezelje a program, normális esetben explicit módon kiszámoljuk és tároljuk.
- *Computation of SMCs*: *sequential*. Itt beállíthatjuk nagy modellek esetén, hogy párhuzamosan több gépen számolja a program a beágyazott Markov lánc független részeit, így kihasználva, hogy nagy aszinkron jellegű Markov láncok analízise jól párhuzamosítható.
- *Precision of arithmetics*-nél tudjuk az integrálás pontosságát megadni, ahol beállíthatunk tetszőleges pontosságot (*arbitrary*) is (ebben a példában bőven elég a *double*).
- Fontos beállítási lehetőség az *Experiment*, ha ezt választjuk, akkor megadhatjuk egy tetszőleges paraméterét (változóját) a rendszernek, amelyet iterációnként növelve az analízist sorban több paraméterrel is lefuttatja a program. Ezt egy későbbi részben megvizsgáljuk.

Az analízis eredménye egy Analysis Output ablakban fog megjelenni, sok más információ mellett:

```
state probabilities:
0.0909091   0.909091
```

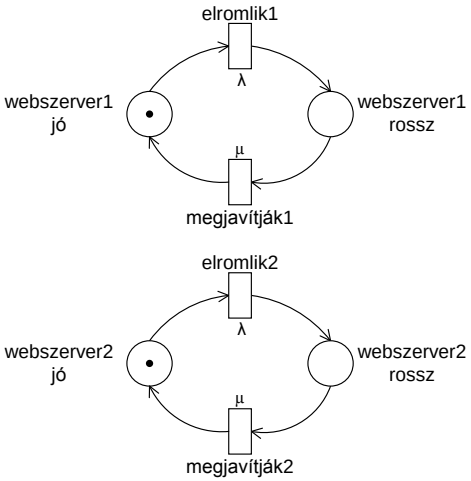
Azaz a webszerver rendelkezésre állása 90,909%.

Megtekinthetjük a *modellnév.dir* könyvtárban a futás során létrehozott fájlokat. Ezek közül számunkra főleg a *.RESULTS* kiterjesztésű lehet érdekes, melynek tartalma általános esetben a tranzíciók áteresztőképességét tartalmazza, de ha bonyolultabb analíziseket futtatunk, akkor annak az eredménye is ide kerül. Ezen kívül érdemes megemlíteni, hogy a *.INV* kiterjesztésű mappába a Petri háló P invariánsai kerülnek kiírásra.

4.2 Módosított, redundáns modell

Az analízist elvégezzük két webszerverre is, hogy modellezzük a redundáns megoldást, ezért a modellt módosítani kell.

A módosított modellben felveszünk még egy webszervert, az így kapott Petri háló a következő ábrán látható.



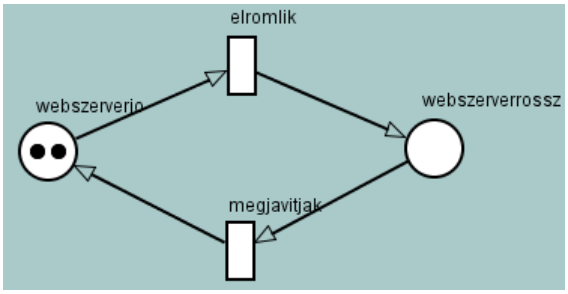
15. ábra: Két webszerver Petri háló modellje

Ekkor az analízis eredményeként a következő jön ki:

```
state probabilities:
0.0826446          0.00826446          0.0826446          0.826446
```

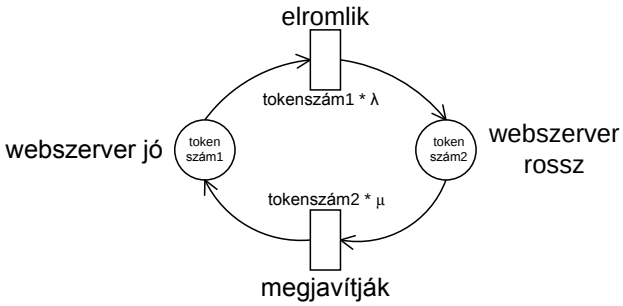
A valószínűségekhez tartozó állapotok: az első és harmadik az, amikor egyik webszerver hibás, a másik jó, a második valószínűség az, amikor mindkét webszerver rossz, és az első az, amikor mindkét webszerverünk tökéletesen funkcionál. Számunkra, mivel redundáns az architektúra, a szolgáltatás akkor áll csak le, ha mindkét webszerver rossz, ennek a valószínűsége állandósult állapotban 0,82%.

Lehetőségünk van a működést azonban máshogy is modellezni: jogos ötlet, hogy tegyünk két tokent a *webszerver jó* helyre, és így modellezzük, hogy két webszerverünk van.



16. ábra: Két webszerver TimeNET modellje

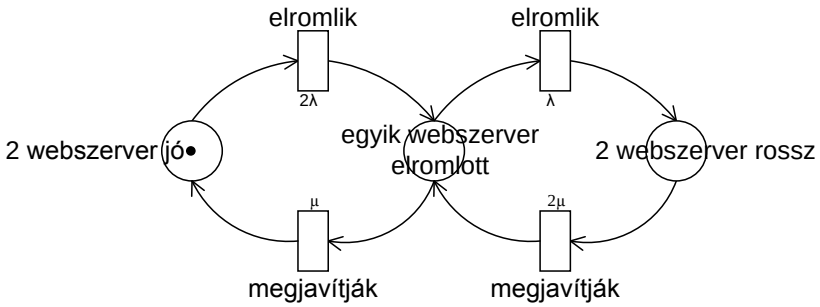
Ez az egyedüli változtatás önmagában azonban kevés, hiszen kezdőállapotból így ugyanakkora a valószínűsége, hogy meghibásodjon egy szerver, mintha csak egy szerverünk lenne (az elromlik tüzelési rátája nem változott). Ezt tudjuk orvosolni azáltal, hogy ha a tüzelési rátákat jelöltségfüggővé (marking függő) alakítjuk, amit a következő ábrán láthatunk:



17. ábra: Jelölésfüggő tüzelési ráták

Így annak az eseménynek az átmeneti intenzitása, hogy a két szerver közül egyik elromlik, kétszer akkora lesz, mint egy szerver esetén, a modell helyesen fog viselkedni. A **TimeNET** eszközben azonban nem lehet explicit módon (tetszőleges) jelölésfüggő tüzelési rátákat definiálni a tranzíciókhoz.

Másik, kevésbé kompakt modell, ha minden állapotnak külön felveszünk helyet:



18. ábra: Két webszerver Petri hálós modellje

Az állandósult állapotbeli analízis során a következő valószínűségeket adja a **TimeNET**:

```
state probabilities:
0.00826446  0.165289    0.826446
```

Azaz a valószínűségek ugyanazok, mint a korábbi esetben.

Azonban a modell mérete arányosan nő a webszerverek számával, ami nagy redundáns rendszerek esetén kényelmetlenné teszi ezt a modellezési megközelítést.

A **TimeNET** eszköz lehetőséget ad, hogy a tranzíciók tüzelési szemantikájának beállításakor „*InfiniteServer*”-t állítsunk be szerver típusnak. Ez azt jelenti, hogy ha egy tranzíció többszörösen engedélyezett, akkor az összes engedélyezett tokenre konkurensen kezdi el kiértékelni a tüzelést (mint erről már korábban volt szó a beállításoknál).

A **TimeNET**-ben a következő tranzíció beállítást alkalmazzuk:

Qualified Name	Value
text	elromlik
delay	20
serverType	InfiniteServer
preemptionPolicy	PRD
DTSPNpriority	1

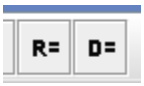
19. ábra: Tranzíció konkurens tüzelés beállítása

Így jól paraméterezhető modellt kapunk, hiszen a tokenszám növelésével egyszerűen tudjuk modellezni a redundáns webszervereket.

4.3 Bonyolultabb kifejezések, mértékek, költség optimalizáció:

A **TimeNET** lehetőséget ad egyéb függvények kiszámolására is. Ehhez definíciókat (**TimeNET**-ben „*definition*”) lehet megadni, továbbá egyéb mértékeket (**TimeNET**-ben „*measure*”) lehet definiálni.

Ezt az alsó panel „**R=**” és „**D=**” gombjára kattintva tehetjük meg.



20. ábra: measure és definition gombok

Ha kiválasztjuk a definíció megadását, akkor egy konstanst tudunk definiálni, melyet a program behelyettesít minden helyre majd, ahol a definícióval hivatkozunk rá. Ez egy kényelmi extra, mely a modell paraméterezhetőségét könnyíti meg. Az alábbi ábrán a tokenszám definíciója látható:

Qualified Name	Value
defType	int
name	token
expression	2

21. ábra: Definition példa

Innentől használható ez a konstans, egyszerűen a nevével hivatkozunk rá. Például, ha szeretnénk beállítani a webszerverek kezdő mennyiségét, azaz a „*webszerver jó*” hely kezdő tokeneloszlását, akkor az *initialMarking* értékét „*token*”-re állítjuk:

Qualified Name	Value
text	webszerverjo
initialMarking	token

22. ábra:tokenszám beállítás

Megvizsgáljuk, hogy hogyan változik a webszerverekből épített rendszerünk elérhetősége az idő függvényében. Ehhez definiálnunk kell egy „*measure*”-t az elérhetőségre, azaz teljesítmény mérőszámot (a referencia általában teljesítmény mérőszámnak nevezi, természetesen ez lehet más jellegű függvény is). Ezt a következőképpen tehetjük meg: a „*measure*” gombra kattintva az alsó panelen, majd a modell bármely részére kattintva megjelenik a szerkesztő ablak, ahol az értékeket megfelelően beállítva létre is hoztuk.

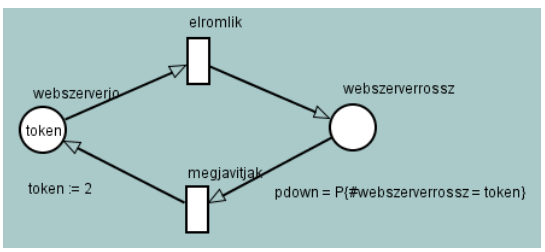
Insert measure

Qualified Name	Value
name	pdown
expression	P{#webszerverrossz = token}
result	

23. ábra: *measure* példa

Az „*expression*” értékére „*P{#webszerverrossz = token}*”-t beállítva kaphatjuk meg a rendszer működésképtelen állapotban tartózkodásának valószínűségét (itt meg kell jegyezni, hogy tapasztalatok alapján az ezzel ekvivalens „*P{#webszerverjo = 0}*” mértékre megbízhatóbban működik a program, főleg a bonyolultabb, „*Experiment*” és „*Transient*” jellegű analíziseknél).

A kapott modell a következő lesz így:



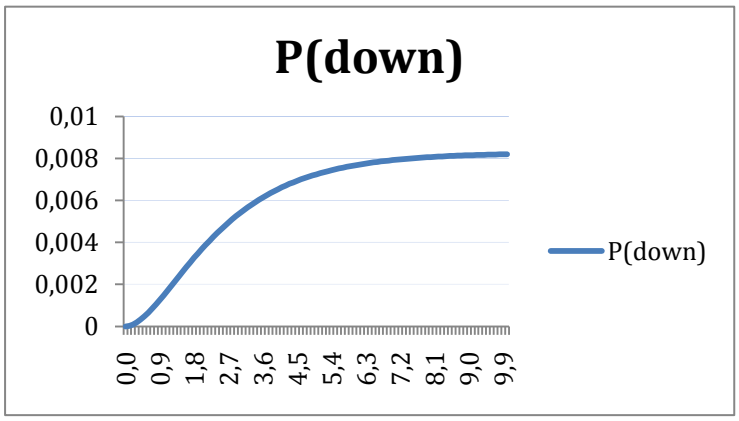
24. ábra: paraméterezett Petri háló

Amennyiben szeretnénk a „*pdown*” értéket vizsgálni, tranziens analízist kell végrehajtanunk. Ezt a főmenüben *Evaluation*-t kiválasztva, majd a legördülő lehetőségek közül *Transient Analysis* pontot kiválasztva tehetjük meg. Itt sok opciót határozhatunk meg az analízis mikéntjére, azonban a program ezeket a változtatásokat legtöbbször (elég gyakran) figyelmen kívül hagyja, és a *default* értékekkel indul el.

A fontosabb futási paraméterek:

- *Transient model time*: tranziens analízis ideje
- *Precision*: kívánt pontosság
- *Output form*: ha a *curve* opciót választjuk, akkor a modellnév.CURVES fájlba kiírja a tranziens futás során vizsgált tulajdonság értékeit, melyet más eszközök (pl: Microsoft Excel) segítségével megjeleníthetünk
- A többi paraméterből többnyire megfelelnek az alapbeállítások.

Az alábbi ábrán a *Transient model time* 10 értékkel elindított futás eredménye látható (korábban definiált „*pdown*” mérték változása az idő függvényében):



25. ábra: Pdown változó idő függvénye

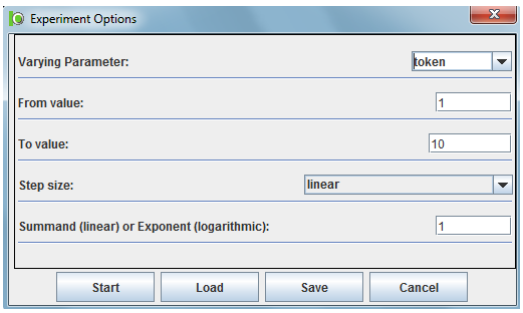
Komplexebb analízisek elvégzésére is képes a program. Lehetőségünk van a definícióként megadott konstans érték függvényeként ábrázolni bizonyos mértékeket.

Tegyük fel, hogy a webszerverek által nyújtott szolgáltatáson a vállalatnak 50 000 000 Ft haszna van. Egy webszerver fenntartása éves szinten 500 000 Ft-ba kerül, és egy nap szolgáltatás-kiesés 1000 000 Ft-ba kerül. Ekkor a következő mértéket definiálhatjuk:

*nyereseg = 50000000-(token*500000+P{#webszerverrossz = token}*365*1000000)*

Ezután az *Evaluation* menü *Stationary Analysis* menüpontjára kattintunk, és a következő ablakban az *Experiment* checkbox-ot bejelöljük, lehetőségünk van a kezdeti tokeneloszlás függvényében kiszámolni a cég várható nyereségét.

A megjelenő *Experiment Options* ablakban beállíthatjuk a paramétereket:



26. ábra:Experiment futás beállításai

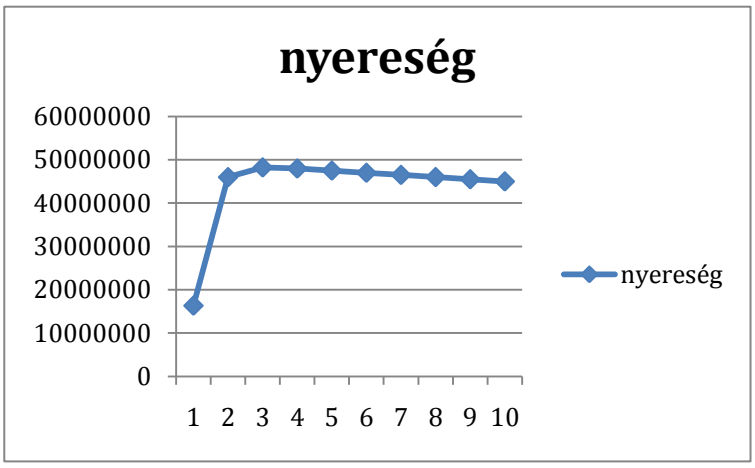
Experiment Options paraméterek:

- *Varying Parameter*: az a definition-ként megadott konstans, aminek az értékét szeretnénk változtatni
- *From value*: kezdeti érték
- *To value*: befejező érték
- *Step size*: beállíthatunk lineáris és logaritmikus lépésközöket
- *Summand (linear) or Exponent (logarithmic):* lineáris esetben a lépésköz, logaritmikus esetben az exponens

Az analízist elindítva a program a *modellnév.EXPRESULT* fájlba írja ki a futás eredményeit, esetünkben ez a következő lesz:

```
# Experiment Results for model 'pelda3a.xml' with varying parameter token
# token     nyereseg
1.0   16318181.8181818
2.0    45983471.0743802
3.0    48225770.0976709
4.0    47975070.0088792
5.0    47497733.6371708
6.0    46999793.9670155
7.0    46499981.2697287
8.0   45999998.2972481
9.0    45499999.8452044
10.0  44999999.9859277
```

A kapott értékeket ábrázolhatjuk külső program segítségével:



27. ábra: Éves nyereség számítása a webszerverek függvényében

Jól látható, hogy az adott költségfaktorok mellett a cég 3 webszerverrel tudja maximalizálni a nyereségét. Az analízist hasonló módon kell elvégezni, ha esetleg többféle szerverünk van, és azokhoz más-más költségek tartoznak.

5 Ellenőrző kérdések

A mérés elején általános, a tárgyhoz tartozó előadások anyagát, és ezen mérési segédlet anyagát érintő ellenőrző kérdéseket kell megválaszolni. Az alábbi felsorolás pár olyan kérdést tartalmaz, amely akár az éles beugróban is előfordulhat:

- Rajzolja fel egy két redundáns SQL szervert, és egy webszervert tartalmazó architektúra Petri háló modelljét!
- Rajzolja fel egy két redundáns SQL szervert, és egy webszervert tartalmazó architektúra RBD modelljét!
- Rajzolja fel egy két redundáns SQL szervert, és egy webszervert tartalmazó architektúra hibafa modelljét!
- Számolja ki azon webszerver rendelkezésre állását, amely elromlásának várható értéke 1 év, megjavításának várható értéke 36,5 nap!
- Mire jó a **TimeNET** eszközben a tranzíciók „*serverType*" beállítása?
- Hogyan számoljuk RBD-k esetén a párhuzamosan kötött komponensek eredő megbízhatóságát a komponensek megbízhatóságából?
- Hogyan számoljuk RBD-k esetén a sorosan kötött komponensek eredő megbízhatóságát a komponensek megbízhatóságából?