Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Nagy Ákos

KÖLTSÉG-OPTIMALIZÁLÁS SZOLGÁLTATÓ-FÜGGETLEN SKÁLÁZÁSSAL A FELHŐBEN TELJESÍTMÉNY-KARAKTERISZTIKÁK ALAPJÁN

KONZULENS

Dr. Kővári Bence

BUDAPEST, 2013

Tartalomjegyzék

Összefoglaló **4**

1 Bevezetés **5**

1.1 Költség-optimalizálás számítási felhőben 5

1.2 A számítási felhő születése; „a feszültségek háborúja” 6

1.3 A számítási felhő ma 8

2 A számítási felhő **11**

2.1 A számítási felhő, mint szolgáltatás-modell 11

2.2 A számítási felhő technológiai szempontból 12

2.2.1 Ki férhet hozzá az erőforrásokhoz? 12

2.2.2 Mihez férhetnek hozzá a felhasználók? 13

2.2.3 Skálázási megoldások 16

2.3 Felhőszolgáltatók 17

2.3.1 Gartner mágikus kvadránsok 17

2.3.2 A szolgáltatók értékelése 18

3 Teljesítménymérés **20**

3.1 Megközelítések 20

3.2 A számítási felhő teljesítményének becslése 21

4 Szolgáltató-független költség-modell **23**

4.1 A szolgáltatók ajánlatainak analízise 23

4.2 Teljesítménymérési algoritmusok 24

4.3 A modell megalkotása 26

4.3.1 A virtuális gépek modellje 26

4.3.2 A tárhely modellje 28

4.3.3 A hálózati erőforrások modellje 29

5 Költségoptimalizálás a modell alapján **30**

5.1 Adatok gyűjtése 30

5.2 Adatok transzformálása és illesztése a modellhez 31

5.3 Költségoptimalizálási algoritmusok 32

5.3.1 Naiv megoldás 32

5.3.2 Lineáris programozás 38

6 Ipari felhasználás és fejlesztési lehetőségek **41**

6.1 Automatikus skálázás 41
6.2 Fejlesztési tervek..... 41
7 Összefoglalás..... 43
Irodalomjegyzék..... 45
A függelék: Azure automatikus skálázás..... 49
B függelék: Enterprise Library Autoscaling Application Block konfiguráció 51
C függelék: A modell virtuális gépeket leíró része..... 53
D függelék: Azure PaaS konfigurációs állomány..... 54
E függelék: Összevonható igények hozzárendelése gépekhez..... 55

Összefoglaló

A számítási felhő (cloud computing) napjaink technológiai fejlődésének egy meghatározó eleme, mely fokozatosan átformálja az alkalmazások fejlesztésének és üzemeletetésének feladatait. A számítási felhő használatának nagy előnye, hogy a felhasználók igény szerint vehetnek igénybe különböző erőforrásokat és mindezért csupán a használatarányos díjat kell megfizetni.

Az igény szerinti erőforrás-felhasználás nagyban megkönnyíti az alkalmazások skálázását. A legtöbb felhőszolgáltatónak van valamilyen kész implementációja a skálázás automatikus megvalósítására, ezek azonban meglehetősen rugalmatlanok. Legtöbb esetben egyszerű, előre definiált, szabály alapú megközelítést alkalmaznak, amik a teljesítménymutatókat egyenként, nem pedig összefüggéseiben figyelik.

A már meglévő skálázási megoldások másik nagy hátránya, hogy erősen szolgáltató-függőek. Ez, és a tény, hogy a nyújtott szolgáltatások és árazásuk nem mindig egyszerűen megfeleltethetőek szolgáltatók között, tovább nehezítik az alkalmazások általános skálázásának kérdését.

Mivel a számlázás a felhasznált erőforrások alapján történik, így a hatékony skálázás kulcsfontosságú. Munkám során egy olyan módszert dolgoztam ki, ami a fenti problémákra ad megoldást. Kidolgoztam egy általános, szolgáltató-független modellt, amivel a szolgáltatók ajánlatai egységesen leírhatóak a jellemző technológiai paraméterekkel és a költségekkel. Ezt követően azonosítottam azokat a teljesítménymutatókat, amik segítségünkre lehetnek az alkalmazások erőforrás-hiányos állapotainak az azonosítására. Az általános szolgáltatásmodellembe négy szolgáltató szolgáltatáscsomagját vettem fel, majd alkalmazások monitorozásával az előzőleg azonosított teljesítménykarakterisztikák mentén adatokat gyűjtöttem. Végül megvalósítottam egy algoritmust, mely az általános modell és a gyűjtött teljesítményadatok alapján képes meghatározni, melyik szolgáltatótól milyen konfiguráció bérlésével elégíthetőek ki az igényeink a lehető legkisebb költségek mellett.

Az algoritmus segít az üzemeltetőknek a költségeket alacsonyan tartani úgy, hogy az alkalmazások zavartalan működése biztosított maradjon. Ezen felül az algoritmus eredményeire építve automatikus skálázó modulok készíthetőek, amelyek emberi beavatkozás nélkül végzik a skálázást.

1 Bevezetés

A dolgozat bevezető fejezetei a dolgozat célját és a megoldandó problémát mutatják be, valamint elővezetik a számítási felhő koncepcióját.

1.1 Költség-optimalizálás számítási felhőben

A számítási felhő a dolgozat egyik alapkövét adja, emiatt érdemes közelebbről megvizsgálni, mi ez, honnan indult és hová tart. A pontos definíciókat és a technológiai áttekintést a dolgozat későbbi fejezetei tartalmazzák, ennek a fejezetnek a célja a dolgozat módszereinek, fogalmainak és eredményeinek elővezetése.

A számítási felhő lehetőséget biztosít a felhasználóknak arra, hogy igényeik szerint foglaljanak le különböző erőforrásokat adott időre egy szolgáltatótól, majd ennek megfelelően fizessenek. Ha az igényeket jól tudják feltérképezni – egyrészt előre megbecsülni, másrészt az erőforrások használata közben monitorozni –, akkor jelentős megtakarításokat érhetnek el az ezekhez legjobban igazodó erőforrás-foglalással. Viszont mind az igények vizsgálata, mind az ez alapján történő erőforrás-optimalizáció egy komplex feladat. A dolgozat legfőbb célja egy olyan módszer kidolgozása, ami különböző, felhőben futó alkalmazásokat monitoroz, meghatározza a mindenkori aktuális igényeket és ezekhez egy olyan konfigurációt, ami költség-hatékonyan képes kielégíteni őket.

A meghatározott módszernek fontos tulajdonsága, hogy az optimális konfigurációt több szolgáltató ajánlatait megvizsgálva alakítja ki. Mivel azonban a szolgáltatók ajánlatai sokszor nehezen összehasonlíthatóak, kialakítottam egy szolgáltató-független modellt, amiben a szolgáltatásokat egységes módon le lehet írni. Ez egyrészt önmagában is hasznos, másrészt az optimalizáció során a döntések meghozatalában segít. A modell alapjául négy felhőszolgáltatót választottam ki, az értékelési szempontok kialakításához pedig ezeknek ajánlatait vizsgáltam meg több tulajdonság alapján (kik használhatják az egyes szolgáltatásokat, pontosan minek a bérlését teszik lehetővé) és meghatároztam azokat a szolgáltatástípusokat, amiket egy tipikus felhőalkalmazás használ (és így a költségekhez hozzájárul).

A modell kialakításához teljesítményteszteket is futtattam, így a dolgozatban az általános mérési megközelítések és a munkám során konkrétan felhasznált algoritmusok is bemutatásra kerülnek.

A modell kialakítása után meghatároztam azokat a teljesítménymetrikákat, amelyek mentén a különböző felhőalkalmazásokat monitoroztam. Ezek a metrikák alkalmasak arra, hogy az alkalmazások erőforrás-igényeit több szempontból leírják.

Utolsóként pedig kidolgoztam két algoritmust, amik a modell alapján meg tudják határozni a metrikákból származtatott igényeknek megfelelő, költség-hatékony konfigurációt (a modellben tartalmazott ajánlatok közül) és értékeltem az algoritmusokat teljesítmény és testreszabhatóság szerint.

1.2 A számítási felhő születése; „a feszültségek háborúja”

A tudomány és technika történetében nem először megy végbe az a folyamat, ami végül a felhő megszületéséhez vezetett; az elektromos áram felfedezése és elterjesztésére tett kísérletek egy hasonló folyamat részei voltak.

Az áram felfedezője, Thomas Edison az egyenáramot preferálta. Az ő találmányai is mind egyenáramra épültek és előállítani is olcsóbb és egyszerűbb, mint a váltakozó áramot. Hátránya azonban, hogy szállítása nem gazdaságos. Mivel a szállításkor a vezetéken elvesző áram magas feszültség és alacsony áramerősség esetén a legkevesebb, így az áramot ilyen állapotba kell hozni, azaz transzformálni a szállítás előtt, aztán pedig az eredeti állapotba visszaállítani. Az egyenáram transzformálása viszont bonyolult, drága eszközöket igényel, amik ráadásul könnyen meghibásodnak így a karbantartásuk további költségeket von maga után. Ezért Edison elképzelése az áram terjesztésével kapcsolatban az volt, hogy aki szeretne áramot használni, annak vásárolnia kell egy generátort és azt az áram felhasználásának helyéhez közel kell ezt üzemeltetnie.

A váltakozó áramot viszont egyszerűbben, tekercsek segítségével lehet transzformálni, így a szállításuk gazdaságossá tehető. Ezt felismerve George Westinghouse (Nikola Tesla szerb fizikus-feltaláló ötletei alapján) – egy másik terjesztési elképzeléssel állt elő. Az ő ötlete az volt, hogy váltakozó áramot nagy mennyiségben, egy telephelyen állítja elő egy szolgáltató, a felhasználók ennél a szolgáltatónál bejelentkeznek, majd a megtermelt áramot elszállítják hozzájuk vezetéken. Így nem kell

generátort venniük és üzemeltetniük, a szolgáltató pedig az elszállított áram mennyiségét kiszámlázza a felhasználóknak.

Végül ez utóbbi szolgáltatás-modell terjedt el – ma leginkább a váltakozó áramot használjuk, egyenáramot már csak néhány dedikált célra használ az ipar (a feszültségek háborújaként is emlegetett versengés egy hosszú időn át tartó csatározás volt; a részletekért lásd [1]).

A számítási felhő születésével kapcsolatban ugyanezt a közművesítési folyamatot szokták megemlíteni. A számítógépek feltalálása után ha valakinek nagy számítási kapacitásra volt szüksége, akkor vennie kellett egy számítógépet („telepítenie egy generátort a felhasználás helyéhez közel”), az új szolgáltatási modellben, a „felhőben” azonban már csak regisztrálni kell egy szolgáltatónál, aki biztosítani tudja ezt a számítási kapacitást. A stabil és szélessávú hálózati elérésnek köszönhetően a távoli menedzsment feladatok, a számítási kapacitás igénybe vétele, lefoglalása és az adatok fel- illetve letöltése (a számítási kapacitás „szállítása”) gazdaságosan és egyszerűen megoldható. Ha a felhasználóknak adott mennyiségű számítási kapacitásra van szükségük adott időre, akkor bérelhetnek egy gépet, ha pedig már nincs szükségük rá, akkor ezt visszaadják. Ha adott mennyiségű tárhelyre van szükségük, akkor az igényeiknek megfelelően bérelhetnek, a szolgáltatók pedig a felhasznált erőforrásoknak megfelelően számláznak a felhasználóknak. Így a felhasználóknak nem kell az üzemeltetési problémákkal foglalkozniuk, a szolgáltatók pedig nagy mennyiségben, olcsóbban elő tudják állítani a számítási kapacitást.

Bár ma még csak a valóban nagy kapacitást igénylő felhasználóknak éri meg erre a modellre áttérni, az ezek által nyújtott szolgáltatásokból már mindnyájan profitálhatunk. Sok olyan felhőszolgáltatás van, amelyek nélkül már az egyszerű felhasználók mindennapjai is elképzelhetetlenek lennének: a Dropbox, a SkyDrive vagy a GMail mind ilyenek, de kisebb weboldalak vagy blogok hostolására is jó alternatíva lehet a felhő, hosszú távon pedig kétség kívül még nagyobb térhódítást várhatunk. A következő fejezet a felhő jelenlegi és várható elterjedtségét mutatja be.

1.3 A számítási felhő ma

Az első számítási felhőt az Amazon cég indította be 2006-ban [2]. Egy hatalmas és globális webes kereskedelmi rendszerről lévén szó, az Amazon soha nem engedhette meg azt, hogy szerverei túlterhelődjenek. Ennek jegyében gépparkjuk mindig a maximális terhelésre volt méretezve. Mivel azonban a terhelésük az év végén ugrik meg, az év nagy részében kapacitásuk szabadon állt – e köré a szabd kapacitás köré szerveztek egy infrastruktúrát, amivel megszületett az első felhőszolgáltatás.

Az elmúlt hét évben a felhőszolgáltatók köre szélesedett, az IT piac legtöbb nagy szereplője rendelkezik egy saját felhőszolgáltatással; az Amazon mellett a Microsoft, a HP, a Google, a Dell, az Oracle mind beléptek a piacra, és emellett kifejezetten felhővel foglalkozó szolgáltatók is megjelentek.

A kínálat kereslet nélkül nem bővülne ilyen mértékben; a következő néhány adat azt mutatja, hogy a számítási felhő valóban népszerű az IT piacon.

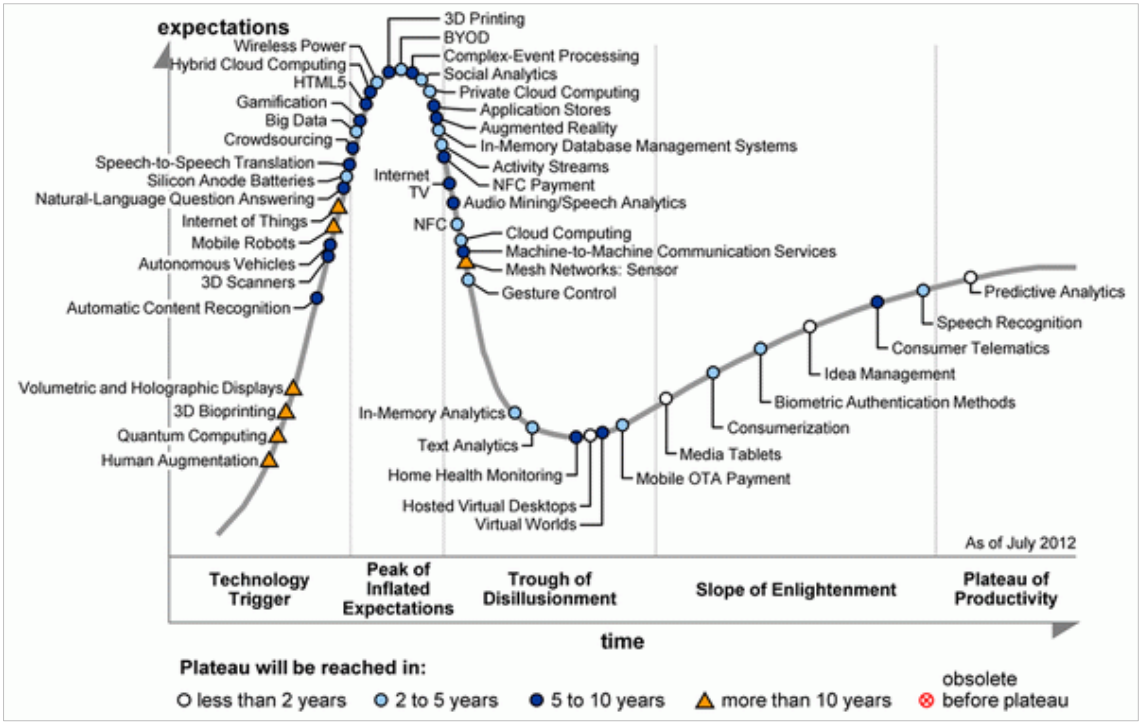
A Rackspace egyik tanulmánya szerint 10-ből 9 IT vezető pozitív véleménnyel van a felhőről [3].

A növekedés folyamatos, a növekedés üteme pedig meggyőző. A Gartner egyik tanulmánya szerint az IT cégek kiszervezésekben tapasztalt növekedés 38%-át az infrastruktúra-szolgáltatások bevezetése adta 2012-ben – 2011-ben ez a szám még csak 8% volt [4]. A Gartner egy másik tanulmánya szerint 2012 végére a publikus felhőszolgáltatásokból származó bevételek meghaladták a 109 milliárd dollárt – ez 91 milliárd dollárral több, mint az azt megelőző évben [5].

Ennek megfelelően a munkáltatók is egyre gyakrabban megkövetelik alkalmazottaiktól a felhő ismeretét. A CompTIA egyik tanulmánya szerint a megkérdezett cégek 60%-a folyamatosan készíti fel az IT részlegek dolgozóit a felhő használatára [6].

Az előrejelzések pedig a jelenlegi számoknál is szignifikánsabb növekedést sejtetnek. Cisco egyik jóslata szerint az elkövetkezendő időben a globális adattárházak 75%-a a felhőbe kerül [7]. Az IDC tanulmánya azt mutatja, hogy 2016-ra a teljes IT világ bevételének 16%-át felhőszolgáltatások díjai fogják adni [8]. A Merill Lynch bank egy kutatása alapján 2020-ra a felhőszolgáltatások piaca 220 milliárd dolláros üzlet lesz [9].

Ezek a számok magukért beszélnek – a számítási felhő már most hatalmas piacot jelent nagy lehetőségekkel, az előrejelzések pedig szintén ígéretesek. Nem csak gazdasági, hanem technológiai szempontból is érdemes egy pillantást vetni a felhőre. A Gartner saját technológiai életciklus-görbéjén 2012-ben a számítási felhő jelenleg abban a fázisban volt, amikor a kezdeti lelkesedés után alábbhagy az érdeklődés („through of disillusionment"). Egyre kevesebb az újdonság, a hirtelen nyereség lehetősége egyre kisebb, a felhőszolgáltatóknak pedig folyamatos innovációkat kell véghezvinni ahhoz, hogy továbbra is megtartsák az eddigi ügyfeleket és továbbiakat szerezzenek. Ez remek időszak arra, hogy új technológiai megoldások szülessenek a felhőre építve.



1. ábra - A Gartner technológiai életciklus görbéje 2012 júliusában [10]

Fontos, hogy ez „kiábrándulás" nem azt jelenti, hogy a felhő visszaszorul (a számok éppen az ellenkezőjét mutatják), csupán azt, hogy kezdenek világossá válni a korlátai és szép lassan kikristályosodik, hogyan és mire érdemes használni. Az életciklus utolsó fázisa, a termelékenységi plató („plateau of productivity") az a szakasz, amikor a technológia tömeges adoptációja elindul – ez a Gartner szerint a felhőnek még 2-5 évébe telik (lásd 1. ábra).

Ezeket és az előző statisztikákat figyelembe véve tehát azok, akik nem várják meg ezt az időszakot, hanem már most (vagy eddig) valamilyen módon elkötelezik magukat a

felhő mellett (akár kutatási, akár fejlesztési vonalon, akár felhasználóként, akár szolgáltatóként), hatalmas előnnyel fognak indulni egy százmilliárd dolláros piacon.

2 A számítási felhő

Az előző fejezetek bevezették a számítási felhő koncepcióját és megmutatták a benne rejlő lehetőségeket. A most következő fejezetek túlmennek a koncepción és a megvalósítást mutatják be különböző szempontokból.

2.1 A számítási felhő, mint szolgáltatás-modell

A számítási felhő fogalmát sokféleképpen lehet értelmezni. A dolgozatban a NIST (National Institute of Standards and Technology; az USA technológiai szabványokkal foglalkozó szervezete) széles körben elfogadott definícióját [11] fogom alkalmazni:

„*A számítási felhő egy modell, amelyben lehetőség van hálózaton keresztül, bárhonnan, kényelmesen, igény szerint konfigurálható számítási erőforrások (pl.: hálózatok, szerverek, tárhely, alkalmazások és szolgáltatások) egy megosztott csoportjához hozzáférni. Az erőforrások lefoglalása és felügyelete minimális menedzsment-erőfeszítéssel és szolgáltatói interakcióval megoldható.*”

A definíció legfontosabb tanulsága, hogy bár az Amazon hívta éltre a számítási felhőt, és bár általában a felhőt mindig technológiai oldalról közelítjük meg (ennek részletes kifejtését lásd egy későbbi fejezetben), **valójában egy modellről van szó.** Ez a modell pedig – ahogy a definíció is kifejti – az alábbiakat teszi lehetővé felhasználók számára:

- **erőforrások igény szerinti lefoglalása:** Ez különbözteti meg elsősorban a számítási felhőt az eddigi hosting megoldásoktól. Lehetőség van arra, hogy a terhelésnek megfelelően alakítsuk a lefoglalt erőforrásokat és ennek megfelelően történik a számlázás is.
- **megosztott erőforrások:** A felhasználók által igénybe vett erőforrások virtuális erőforrások. A színfalak mögött ezeket az erőforrásokat természetesen egy fizikai erőforráshalmaz támogatja, ám ehhez a virtualizáció révén megosztva férnek hozzá a felhasználók. Ez a virtualizáció adja a felhő gazdaságosságának és konfigurálhatóságának alapjait.
- **minimális szolgáltatói beavatkozás:** Ez egyrészt jelenti azt, hogy az erőforrásokat mindenki saját magának tudja lefoglalni a felhőben, nincs szükség

a szolgáltatónál kérvényekre, engedélyekre, beszerzésre. Másrészt pedig azt jelenti, hogy a fizikai infrastruktúra meghibásodott elemeit a számítási felhő autonóm módon kikapcsolja a rendszerből úgy, hogy közben garantálja az erre épülő virtuális erőforrások zavartalan működését.

Fontos, hogy a modell nagyon sok mindenre nem tér ki. Az erőforrások közül néhánynak a virtualizációja egyértelmű; ha például tárhelyre van szükségünk, akkor igényeinknek megfelelően foglalhatunk. Mi a helyzet azonban mondjuk egy webszerver esetén? Kapunk egy kész virtuális gépet processzorral, memóriával, tárhellyel? Vagy csak processzor és memória van a gépben, hozzá külön kell foglalni virtuális tárhelyet igény szerint? Van-e rajta operációs rendszer? Van-e rajta esetleg webszerver felkonfigurálva? Ki végzi a szoftver frissítését? És egyáltalán kik vehetik igénybe a szolgáltatásokat? Ilyen és hasonló kérdések merülnek fel egyéb erőforrásokkal kapcsolatban is. Ezeknek a megválaszolása pedig nem egyszerű, eltérő feladatok eltérő igényeket vetnek fel – ennek megfelelően vannak különböző szolgáltatás(csoport)ok, amelyek ezekre a kérdésekre eltérő technológiai implementációval válaszolnak.

2.2 A számítási felhő technológiai szempontból

Az előző fejezet végén bevezetett kérdésekre a különböző, konkrét szolgáltatásfélék különböző módokon válaszolnak. Ezek alapján különböző csoportosításai alakulnak ki az elérhető felhőszolgáltatásoknak; a következő fejezetek ezeket mutatják be.

2.2.1 Ki férhet hozzá az erőforrásokhoz?

Az alapján, hogy ki férhet hozzá egy szolgáltató erőforrásaihoz (vagy más megközelítésben, de ehhez szorosan kapcsolódóan: hogy ki a szolgáltató), három különböző típusú felhőt szokás megkülönböztetni: publikus, privát és hibrid felhőt.

A **publikus felhő** az, amire legtöbben gondolnak a számítási felhő kapcsán. Ezekhez a felhőkhöz mindenkinek van hozzáférése, csupán regisztrálnia kell magát a szolgáltatónál, megadni a számlázási adatokat, és már lehet is használni a felhő erőforrásait. Ilyen felhőszolgáltatások a Microsoft Azure, az Amazon Elastic Cloud Computing (EC2), a Rackspace és a legtöbb, sokak számára is ismert szolgáltatás.

A **privát felhő** lényegében ennek az ellentéte; ennek a felhőnek az erőforrásaihoz csak egy bizonyos felhasználói kör férhet hozzá. Tipikus példája egy ilyen

infrastruktúrának az, amikor egy cég saját, erős fizikai IT infrastruktúrával rendelkezik, és a cég egyes részlegei az igényeiknek megfelelően foglalhatnak le az erre épülő virtuális erőforrásokból.

Bár a legtöbb felhőszolgáltató publikus felhőt működtet, nagyon soknak van megoldása arra, hogy privát felhőt építsenek ki az infrastruktúrájuk fölé a vállalatok (az Amazon, az Azure és a Rackspace is alkalmas privát felhő kialakítására is). A VMWare és az Oracle megoldásai kifejezetten privát felhők kialakítását segítik elő.

A két megoldás kombinációja a **hibrid felhő**. Ez egy olyan felhő, aminek egy része a publikus felhőben, egy része pedig privát felhőben üzemel; ez ideális lehet, ha a felhasználó át akar térni felhő alapú szolgáltatásokra, de a saját, fizikai infrastruktúráját is meg akarja tartani.

Mivel jellegéből adódóan a publikus felhőben végzett mérések adnak a legjobban általánosítható eredményeket, a dolgozat a különböző szolgáltatók publikus felhő megoldásait vizsgálja.

2.2.2 Mihez férhetnek hozzá a felhasználók?

Az alapján is meg tudjuk közelíteni a felhőszolgáltatásokat, hogy hol végződik a szolgáltató felelőssége a rendszer karbantartásában és működtetésében és hol kezdődnek a felhasználó feladatai.

Ha a virtualizált erőforrások lényegében csak a fizikai erőforrások egyszerű leképzései, akkor infrastruktúra-szolgáltatásokról (Infrastructure-as-a-Service, IaaS) beszélünk. Ebben az esetben például egy virtuális gép bérlése esetén csak a virtuális infrastruktúra mögötti fizikai infrastruktúra karbantartása a szolgáltató feladata. A virtuális gép operációs rendszerének beállítása, karbantartása, frissítése, a megfelelő szoftverek telepítése és beállítása mind a felhasználó feladata. Ezzel a szinttel tipikusan az üzemeltetők dolgoznak – a gépeket tetszőlegesen be tudják konfigurálni, a hálózati beállításokat tetszőlegesen meg tudják adni. Bár ma már szinte minden felhőszolgáltatónak vannak infrastruktúra-szolgáltatásai, ennek iskolapéldája az Amazon szolgáltatása – indulásakor kifejezetten infrastruktúra-szolgáltatás volt, és még ma is ez az elsődleges profilja.

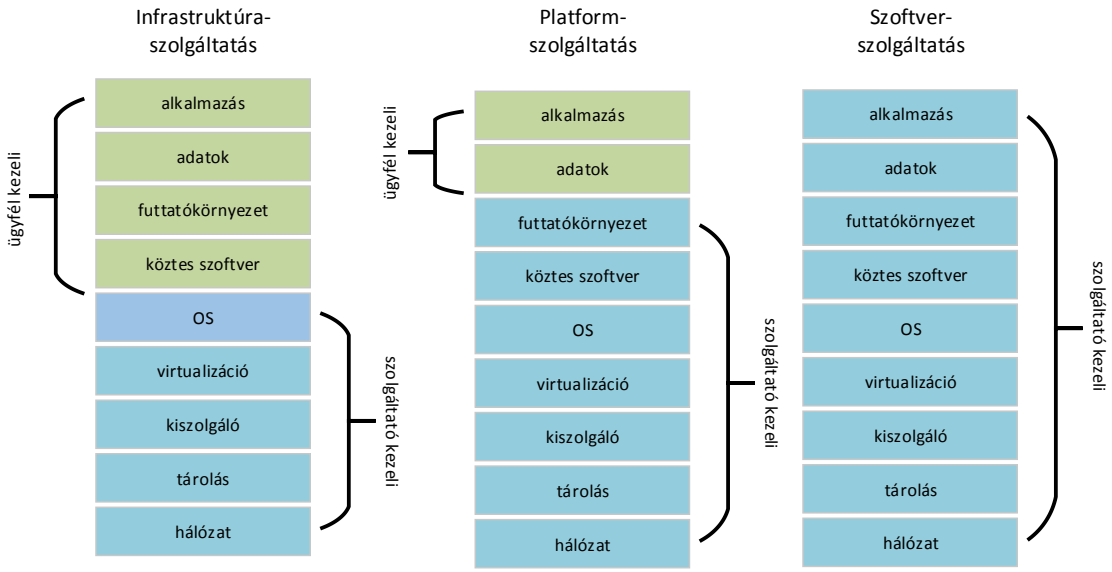
Eggyel magasabb absztrakciós szinten a felhasználók kész platformot kapnak a szolgáltatótól. Ebben az esetben például egy feltelepített számítógépet kap a felhasználó,

bekonfigurálva, telepítve a megfelelő szoftverekkel (például Windows operációs rendszer .NET platformmal és IIS-sel vagy Ubuntu operációs rendszer Apache webszerverrel és php futtatókörnyezettel, mindezek megfelelően beállítva), az operációs rendszer és a többi szoftver karbantartása és frissítése a szolgáltató feladata. Ezzel a szinttel tipikusan a fejlesztők dolgoznak – csak meg kell írniuk az alkalmazást és telepíteni a felhőbe (erre a megfelelő platformok népszerű fejlesztőkörnyezetei általában tartalmaznak kényelmes, beépített lehetőségeket), nem kell foglalkozniuk az üzemeltetési kérdésekkel (a Twitter sikertörténetében is fordulópont volt, amikor eredeti fejlesztője, Jason Gandron áttért erre az üzemeltetési megoldásra [12]). Mivel itt a felhasználók kész platformot kapnak felkonfigurálva, így ezt a modellt platform-szolgáltatásoknak (Platform-as-a-Service, PaaS) nevezzük. Bár a legtöbb szolgáltatónak szintén vannak platform-szolgáltatásai is, ennek iskolapéldája a Windows Azure, amely bár szintén ajánl infrastruktúra-szolgáltatásokat is, mégis elsődleges profilja a platform-szolgáltatások.

A harmadik megoldásban nem csak platformot, hanem kész szoftvert kapnak a felhasználók; ezekre szoftver-szolgáltatásként (Software-as-a-Service, SaaS) hivatkozunk. Ennek jó példái a Microsoft Office 365 szolgáltatása, amely az Office irodai szoftvercsomag felhős verzióját nyújtja, vagy a Team Foundation Services, amely egy teljes, csapatmunkát, verziókövetést és szoftveréletciklus-menedzsmentet megvalósító eszközt ad a felhasználók kezébe (lényegében a Team Foundation Server felhőbe kihelyezett verziója) felhő alapon. Ezek közé a szolgáltatások közé sorolják általában a GMailt, a Yahoo-t vagy gyakran a Facebookot is.

Ennek a megoldásnak az előnyei is egyértelműek – mivel a szoftver teljes karbantartása a szolgáltató feladata, így nem kell például az irodai szoftverek telepítésével, frissítésével a rendszergazdákat terhelni.

Ahogy a 2. ábra is mutatja, ezek az absztrakciós szintek egymásra épülnek:



2. ábra – IaaS, PaaS és SaaS szolgáltatások [13]

Az egymásra épülést jól példázza az Amazon vagy az Azure felhőszolgáltatásainak a fejlődése. Az Amazon először csupán infrastruktúra-szolgáltatásokat biztosított, viszont erre a rétegre építve most már ajánl platform-szolgáltatásokat is. Az Azure szolgáltatásainak fejlődése kétirányú volt; elsőként csupán platform-szolgáltatásokat nyújtott, viszont szép lassan a Microsoft elkezdte kiajánlani az e mögött megbújó infrastruktúrát is, valamint ezzel párhuzamosan a platformra saját szoftvereinek némileg módosított verzióit telepítette, így nyújtva szoftver-szolgáltatásokat is.

Sok szolgáltatónak ezek mellett vannak adott feladatra dedikált megoldásai is (médiasugárzás, webhosting, „nagy adat” feldolgozás), ezek viszont erősen szolgáltató-specifikusak, nehezen írhatóak le általánosan, így a dolgozatnak nem célja ezek bemutatása.

A dolgozat az IaaS megoldásokat vizsgálja, mivel azonban a PaaS megoldások legtöbbször erre közvetlenül építenek a szolgáltatóknál, így a költség-optimalizálási rendszer prototípusában erről az oldalról kerül megvalósításra, nagyban megkönnyítve ezzel a mérést.

2.2.3 Skálázási megoldások

Hogy segítsenek kihasználni az igény szerinti hozzáférésben rejlő megtakarítási lehetőségeket, néhány szolgáltató ajánl automatikus skálázási megoldásokat. Ezeknek segítségével az erőforrások külön beavatkozás nélkül, a felhőben futó alkalmazások igényeinek megfelelően kerülnek lefoglalásra a szolgáltató által. Ennek lehetséges megközelítéseit mutatja be ez a fejezet az Azure által nyújtott lehetőségeken keresztül.

Az egyik megoldásban a processzor-kihasználtság és az üzenetsorban lévő üzenetek száma alapján skálázhatóak a gépek, valamint megadható egy minimum gépszám és egy maximum gépszám, amely határokat nem szabad túllépni skálázáskor. Ezeket a skálázási szabályokat időszakokba tudjuk rendelni és különböző időszakoknak más-más szabályokat megadni. Ez a módszer a webes felügyeleti portálon keresztül érhető el (képernyőképekért lásd az A függeléket). A megoldás nem túl rugalmas, csupán alapvető skálázási feladatokat tud ellátni.

Egy másik megközelítés az Enterprise Library Autoscaling Application Block skálázási lehetőségeinek felhasználása [14]. Ez a megoldás is szabályalapú megközelítést használ, viszont jóval rugalmasabb az előzőnél. Két fajta szabálytípust definiál a keretrendszer:

- megkötések: Megadható, hogy bizonyos időszakokban mennyi lehet a minimum vagy maximum példány, ami kiszolgálja az adott alkalmazást.
- reaktív szabályok: Ezek a szabályok adják meg, hogy milyen skálázási akciókat kell végrehajtani.

A kétfajta szabálytípus önmagában még nem teszi sokkal rugalmasabbá ennek az osztálykönyvtárnak a használatát. Valódi előnye ennek a megoldásnak az, hogy a mérőszámok, ami alapján a skálázás történik, sokkal jobban testre szabhatóak, lényegében a Windows teljesítményszámláló-megoldása van kiajánlva ebben az API-ban. Az egyes teljesítményszámlálókhoz lehetőség van különböző aggregációs műveletek beállítására is (például hogy átlagot mérjen bizonyos időszakra vagy a maximumát vegye az értékeknek), majd ha a mért érték meghalad egy beállított határértéket, akkor végrehajtódik a skálázási akció. Szintén előnye ennek a megoldásnak, hogy a szabályoknak „erősséget” is beállíthatunk, így ütköző szabályok esetén az erősebb hatása érvényesül (részletes példáért lásd az B függeléket).

Az Amazon automatikus skálázása is az Autoscaling Application Block megoldásához hasonló. A CloudWatch szolgáltatás segítségével valósítható meg a gépek felügyelete, erre a felügyeleti rendszerre építve pedig skálázási szabályokat adhatunk meg [15].

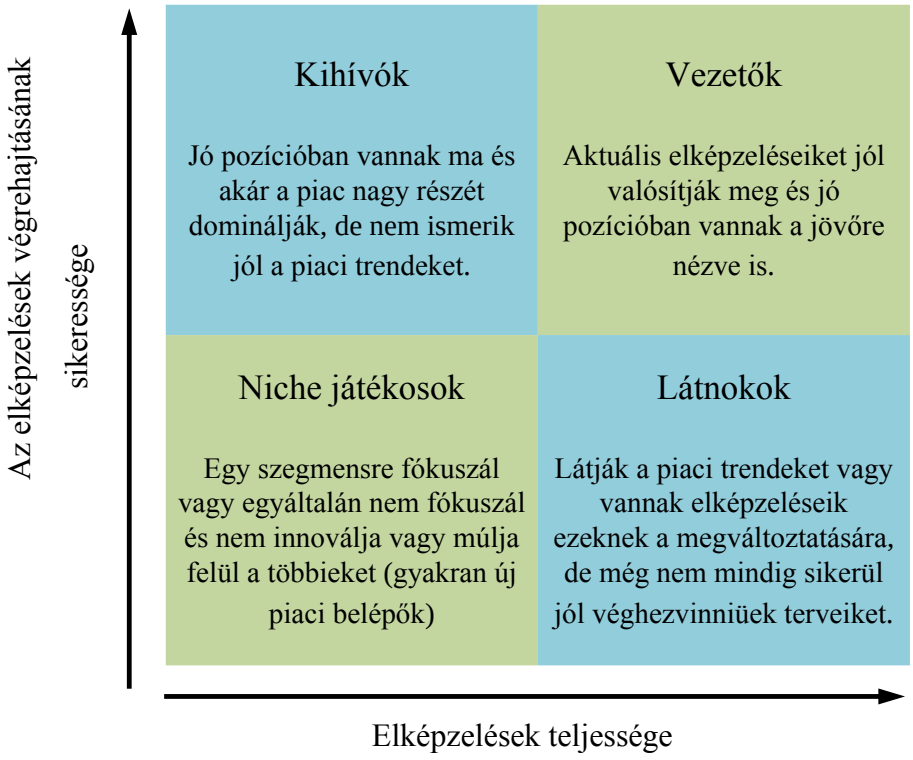
Ezeknek a skálázási megközelítéseknek azonban nagy hátránya, hogy minden szabály csupán egyetlen mérőszám alapján végez skálázást, és bár a mérőszámok aggregálhatóak, a mérőszámok közötti összefüggések nem kerülnek figyelembe vételre. Az általam kidolgozott módszer erre a hiányosságra ad megoldást.

2.3 Felhőszolgáltatók

Ebben a fejezetben bevezetésre kerülnek azok a felhőszolgáltatók, amelyeket a dolgozatban bemutatott általános, szolgáltató-független modell megalkotásakor megvizsgáltam. Ehhez a Gartner piackutató cég mágikus kvadránsok (magic quadrants) elemzését használom fel.

2.3.1 Gartner mágikus kvadránsok

Bár a dolgozatnak nem célja, hogy gazdasági elemzéseket részleteiben bemutasson, a felhőszolgáltatók képességeit a Gartner piaci elemző cég egyik módszere nagyon jól szemlélteti. Az elemzési módszer a szolgáltatókat az két dimenzió mentén négy kategóriába osztják:

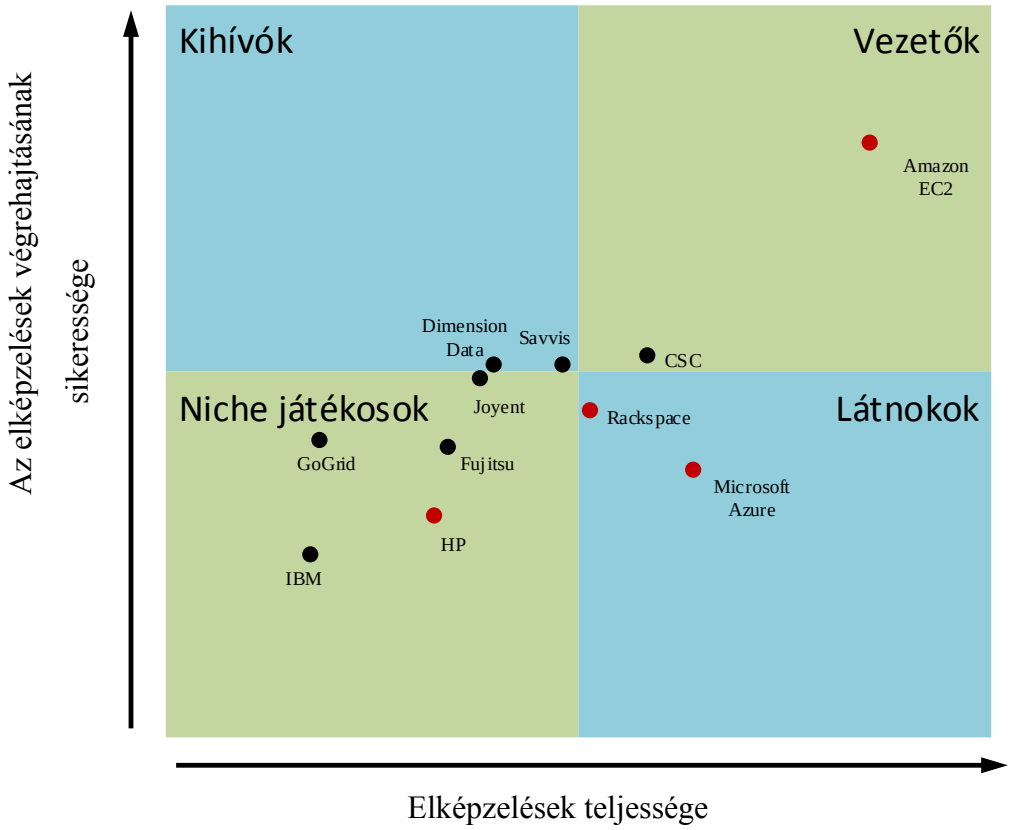


3. ábra - A Gartner mágikus kvadránsok elemzési módszere

A 3. ábra az egyes kategóriákba tartozó szolgáltatók általános jellemvonásait mutatja be. A mágikus kvadránsokat elemzési módszert részletesebben [16] mutatja be.

2.3.2 A szolgáltatók értékelése

A 2013 augusztusában kiadott elemzés alapján az IaaS szolgáltatók mágikus kvadránsai a következőképpen alakulnak (részlet, a teljes kimutatást lásd [17]):



4. ábra – IaaS szolgáltatók mágikus kvadránsa

A dolgozatban ezek közül az Amazon, Microsoft (Azure), Rackspace és HP szolgáltatásokat használtam fel a modell felépítésére. Ezekkel a szolgáltatókkal a négy kvadránsból hármat lefedtem; a hiányzó kvadráns, a „Kihívók” pedig a legtöbb esetben nem jól képviselik a piaci trendeket (lásd 3. ábra).

3 Teljesítménymérés

A processzorok teljesítményének lemérése egy sok szempontból nehéz, összetett feladat. Az órajel frekvenciája (a processzor megahertz értéke) egy gyakran használt érték a teljesítmény leírására, de ez önmagában nem ad valóban használható, összehasonlítás alapjául szolgáló metrikát (ez a probléma az irodalomban megahertz-mítoszként ismert [18]). Az órajel mellett például az utasítások granularitása (RISC vagy CISC architektúra), a processzorokban használt csővezetékek szerveződése is meghatározzák azt, hogy az utasításokkal milyen gyorsan végez egy-egy gép.

A többmagos processzorok elterjedése újabb problémát vetett fel a teljesítmény összehasonlításának kérdésében; egy kétmagos, 3 GHz órajelű processzor minek felel meg? Egy egymagos, 6 GHz órajelű gépnek? Vagy két egymagos, egyenként 1,5 GHz órajelű gépnek? Vagy valójában egyiknek sem?

A most következő fejezet bemutatja a processzorok teljesítménymérésének alapvető megközelítéseit, majd egy későbbi fejezetben, a modell felépítésének tárgyalásakor röviden bemutatásra kerülnek a konkrét implementációk is.

3.1 Megközelítések

A processzorok teljesítményének mérése alapvetően három különböző módszer létezik:

- szintetikus mérések: A természetes nyelvekben a szavak gyakorisága eltérő. Amikor egy idegen nyelvet tanulunk, a nyelvkönyvek első fejezetei tipikusan olyan alapvető konstrukciókat mutatnak be, mint a személyes névmások vagy a létigék használata. Ehhez hasonlóan a programnyelvek utasításai is különböző gyakorisággal fordulnak elő – több program elemzésével felállíthatóak statisztikák arra, hogy milyen utasítások milyen gyakran fordulnak elő. Az utasítások gyakorisága alapján pedig felépíthetőek olyan programok, amik ennek a statisztikának megfelelően tartalmazzák az utasításokat, így lényegében az adott programnyelven írt összes program egy reprezentatív mintáját adva. Ezt a programot lefuttatva különböző processzorokon, jó alapot kapunk a processzorok összehasonlítására, de a megoldás mestersége mivolta kérdéseket vethet fel, ráadásul egy ilyen mérési keretrendszer felépítése időigényes munka.

- valós mérések: Ha adott programot szeretnénk futtatni, akkor ezt a programot több processzoron lefuttatva, majd az eredményeket összehasonlítva egyszerűen ki tudjuk választani a legjobb processzort (természetesen a megfelelő mérési keretrendszerről gondoskodnunk kell). Ennek a megoldásnak előnye, hogy a mérési keretrendszerben használandó program adott, de mivel ez a program nem reprezentatív (hiszen általában adott célra vannak kifejlesztve), így az eredmények nem általánosíthatóak.
- derivatív mérések: Ez a mérési módszer a két előző módszert kombinálja. A valós programokban használt gyakori, bonyolultabb konstrukciókat (pl.: rendezés, keresés) tartalmazza a mérési keretrendszerben felhasznált program – így egyszerre közelebb áll a mindennapokban felhasznált programokhoz és ad valamilyen szinten reprezentatív mintát.

A dolgozatban felhasznált mérési keretrendszerek szintetikus és derivatív mérési keretrendszereket használnak.

3.2 A számítási felhő teljesítményének becslése

A számítási felhő megjelenése szinte azonnal felvetette a kérdést, hogy ezek a rendszerek milyen teljesítménnyel bírnak az eddigi szuperszámítógépekhez vagy grid rendszerekhez képest.

A [19] írói az Amazon EC2 szolgáltatását mérték és hasonlították össze a TOP500 lista[1] gépeivel. Bár akkor még arra a következtetésre jutottak, hogy nem éri meg áttérni a szuperszámítógépekről erre a felhőszolgáltatásra, láttak lehetőséget a rendszerben hosszútávon.

Egy másik munka ([20]) szintén az Amazon EC2 rendszerén végzett teljesítménymméréseket mutat be, kifejezetten nagy számításigényű feladatokat (High Performance Computing, HPC) futtatva. A cikk következtetése, hogy nagy variancia van a szolgáltatás teljesítményében a feladatoktól függően, ennek okaként pedig a virtuális

[1] A TOP500 projekt (http://www.top500.org) célja a világ 500 legerősebb szuperszámítógépének folyamatos nyilvántartása.

környezet megosztott jellegét, a hálózati kommunikációt és a fizikai rendszerben lévő komponensek közti különbségeket azonosítják.

Egy további munka szerzői beazonosították azokat a tudományos feladatokat, amelyek megoldásában különösen jól teljesít az Amazon EC2, és közben egy nagyon sokrétű értékelését állítják fel ennek a rendszernek [21].

Az itt felsorolt néhány munka célját tekintve más, mint a dolgozatomé, viszont a felhasznált módszer hasonló. Szintén az adott szolgáltató (Amazon) publikus infrastruktúra-szolgáltatásának virtuális gépein végez méréseket különböző szintetikus vagy derivatív módszerekkel. A mérések során elsősorban a Linpack programot használták az előző szerzők; ez egyike volt az általam is lefutatott teszteknek (a részletes leírást lásd a 4.2 fejezetben).

Az itt bemutatott munkák elsősorban számítási kapacitással kapcsolatban próbálnak kérdésekre válaszolni, így főleg a virtuális gépek kerülnek megvizsgálásra. Mivel az én célom a költség-optimalizálás (és ennek csak egy eleme a virtuális gépek által elhasznált idő), így én a szolgáltatók infrastruktúra-szolgáltatásainak több elemét is megvizsgáltam.

4 Szolgáltató-független költség-modell

A dolgozat célja, a szolgáltató-független költség-optimalizálás felé az első lépés egy olyan modell megalkotása, amely segít leírni a szolgáltatók ajánlatait egységes módon, így összehasonlíthatóvá téve őket. Ennek a modellnek a kidolgozását mutatják be a következő fejezetek.

4.1 A szolgáltatók ajánlatainak analízise

Az első lépés a vizsgálni kívánt szolgáltatók beazonosítása volt. A kutatásnak ebben a fázisában a már említett négy szolgáltató, az Amazon, a Microsoft, a HP és a Rackspace került kiválasztásra (lásd 2.3.2).

A következő lépés az, hogy eldöntsük, publikus, privát vagy hibrid felhő megoldásokat szeretnénk vizsgálni. Mivel a lehető legáltalánosabb megoldást szerettem volna kialakítani, így a publikus felhőt választottam.

Ezután az egyszerű lépés után egy jóval nagyobb megfontolást igénylő feladat következik – miután a „ki férhet hozzá?” kérdésre válaszoltunk, válaszolni kell a „mihez férünk hozzá?” kérdésre, praktikusan tehát hogy IaaS, PaaS vagy SaaS, esetleg egyéb, adott célre dedikált szolgáltatásokat vonunk be a mérésekbe. Az IaaS mérések mellett két érv szól:

1. A PaaS és SaaS megoldások nagyon eltérőek a négy szolgáltató esetében. A HP esetében például még az IaaS szolgáltatás sok eleme is bétában van a dolgozat írásakor, ez fokozottan igaz a PaaS megoldásra, SaaS megoldása pedig nincs is minden vizsgált szolgáltatónak. Az egyéb szolgáltatások pedig annyira eltérnek szolgáltatónként, hogy általános modellt egyáltalán nem lehet felépíteni rá már ennek a négy szolgáltatónak a vizsgálatakor sem.
2. Mivel a szolgáltatók a PaaS megoldásaikat az IaaS megoldásaikra, a SaaS ajánlataikat és egyéb szolgáltatásaikat pedig vagy közvetlenül az IaaS megoldásokra, vagy a PaaS megoldásaikra építik, így egyrészt megfelelő körültekintéssel ezekre vonatkozóan is levonhatunk következtetéseket az IaaS mérésekből, másrészt pedig megfelelően kialakított mérési keretrendszerekben a méréseket akár PaaS irányból is megközelíthetőek.

Első megkötésem tehát így hangzik:

A kiválasztott négy felhőszolgáltató – Amazon, Microsoft, Rackspace és HP – publikus felhőszolgáltatásának infrastruktúra-szolgáltatásait foglalom bele a modellbe.

Ezután még mindig meg kell vizsgálni az infrastruktúra-szolgáltatásokat részletesen és eldönteni, melyek kerülnek bele a modellbe. Itt három dolgot kell szem előtt tartani:

1. Azokat a szolgáltatásokat mindenképpen bele kell venni a modellbe, amit a legtöbb alkalmazás használ.
2. Azokat a szolgáltatásokat, amik IaaS szolgáltatások, de szolgáltató-specifikusak vagy nehezen általánosíthatóak, nem vehetőek bele a modellbe.

Tovább pontosítva tehát a modell felépítésekor figyelembe vett szolgáltatásokat, a következőket állapítottam meg:

A kiválasztott négy felhőszolgáltató – Amazon, Microsoft, Rackspace és HP – publikus felhőszolgáltatásának infrastruktúra-szolgáltatásai közül a virtuális gépek, a tárhely, az általános hálózati és a tartalomelosztási (content delivery network, CDN) szolgáltatásokat foglalom bele a modellembe.

Fontos még, hogy szolgáltatónként eltérhetnek ezek a megoldások is részleteiben, emiatt a különböző szolgáltatók ajánlatait a különböző technológiai paraméterek mentén is össze kell hasonlítani, mielőtt a modellbe kerülnek.

4.2 Teljesítménymérési algoritmusok

Az előző fejezetben megállapításra került, hogy a virtuális gépeket nyújtó szolgáltatások részei lesznek a modellnek. A virtuális gépeket a bennük lévő memória és processzor írja le a modellben (lokális tárhely egyes szolgáltatóknál nem érhető el közvetlenül a gépekhez, csak a dedikált tárhely-szolgáltatás keretein belül lehet foglalni hozzá háttértárat; ennek elhagyása viszont nem befolyásolja nagymértékben a modellt, mert a legtöbb esetben a rendelkezésre álló háttértár a virtuális gépekben a tárhely-szolgáltatás keretein belül viszonylag olcsón megvehető).

A memória megfelelően jellemezhető a kapacitásával (a gyakorlatban a memória sebessége ritkán szűk keresztmetszet az alkalmazásokban), ahogy viszont az a 3 fejezetben már tárgyalásra került, a processzorok teljesítménye nem írható le egyszerűen egy technikai paraméterrel – ehhez megfelelően megtervezett méréseket kell

végrehajtani. Ennek lehetséges elméleti megközelítéseit a 3.1 fejezet mutatta be, most pedig a konkrét, a modell megalkotásához is felhasznált implementációk kerülnek bemutatásra nagyon röviden.

- **Whetsone**: Szintetikus mérési keretrendszer, amelynek alapjait még Fortran nyelven készült programok elemzésével állították elő a ’70-es években. Azóta is nagyon népszerű mérési keretrendszer, előszeretettel használják processzorok leírására. Ezt a mérést az időt mérve futtattam 1000000 iterációval és ennek mértem az idejét, tehát ebben az esetben a kisebb értékek a jobbak [22].
- **Dhrystone**: Szintetikus mérési keretrendszer, a Whetstone alapján készült, de elsősorban az egészekkel történő műveletek elvégzésére fókuszál. Ez a benchmark MIPS-ben, azaz az 1 másodperc alatt elvégzett millió utasítások számában adja meg a processzor teljesítményét, ebben az esetben tehát a nagyobb értékek a kedvezőbbek [23].
- **Linpack**: Besorolástól függően szintetikus vagy (sokkal inkább) derivatív mérési keretrendszer. Eredeti célja a szoftvernek az volt, hogy lineáris egyenletrendszerek megoldásához szükséges időt hozzávetőleges ki tudják számolni különböző processzorokon a bemenet függvényében, majd ezt egy táblázat formájában felhasználók rendelkezésére tudják bocsájtani. Ennek megfelelően adott méretű lineáris egyenletrendszereket old meg LU dekompozícióval. Ezt a mérést egy 7000x7000 méretű problémán futtattam és az időt mértem, tehát ebben az esetben a kisebb értékek a jobbak [24].
- **Livermore**: Besorolástól függően szintetikus vagy (sokkal inkább) derivatív mérési keretrendszer, amely fizikai és komplex matematikai számításokat végez (például differenciálszámítás, Plank-eloszlás, állapotegyenlet-számítás) Ez MFLOP-ban, azaz egy másodperc alatt elvégzett lebegőpontos műveletek számában adja meg az eredményt, tehát a nagyobb értékek a kedvezőbbek [25].
- **DBench**: Ez a keretrendszer a saját implementációm, derivatív módszert követ. Az alapötlet a ’90-es évek közepén a BYTE magazinban publikált derivatív mérési keretrendszerből származik [26], néhány alapvető algoritmust futtat: prímkeresés, egészek és karakterláncok rendezése kupacrendezéssel, Huffmann-kódolás természetes szövegen (néhány népszerű regény egy-egy részlete) valamint a feladat-hozzárendelési problémát oldja meg magyar módszerrel.

Ezeket mind az időt mérve futtattam adott paraméterekkel (1200000. prím naiv megkeresése, 1,6 megabájtos fájl átkódolása, 30 darab 250000 elemű egésztömb és 10000 darab 100 elemű string tömb rendezése), minden esetben az időt mérve, tehát az alacsonyabb értékek a kedvezőek.

A keretrendszerek mind C vagy C++ programozási nyelven lettek implementálva, hogy szükség esetén a platform-függetlenség biztosítható legyen. A méréseket ezekkel a keretrendszerekkel, Windows Server 2008 R2 operációs rendszereken végeztem el, majd az egyenként kapott értékeket normalizálva és aggregálva alakult ki egy-egy virtuális gép processzorának mérési eredménye.

4.3 A modell megalkotása

Az előző fejezetekben leírtak alapján készült el a végső modell, amelyet az alábbi táblázatok és diagramok szemléltetnek.

4.3.1 A virtuális gépek modellje

A virtuális gépek modellje tehát az IaaS szolgáltatások gépeit írja le az előzőleg meghatározott paramétereikkel: memória mérete valamint a processzor teljesítménymérésével kapott érték.

A modell ezen darabjának felépítésekor két újabb megkötést tettem:

1. A legtöbb szolgáltató virtuális gépei között van egy olyan, amely nem saját processzormaggal fut, hanem több más felhasználó gépeivel közösen, megosztva használ egyet. Mivel az ilyen gépek teljesítménye nagyban függ attól, hogy a többi felhasználó éppen milyen alkalmazásokat futtat a saját virtuális gépén (vagy hogy éppen milyen fizikai gép szolgálja ki a virtuális gépet), így ezek mérése nem tud megbízható eredményt adni – ezeket a virtuális géptípusokat tehát kizártam a modellből.
2. A legtöbb szolgáltató a saját gépeinek teljesítményét általában egy saját referencia-processzor alapján adja meg, például a HP esetében 1 HP Cloud Computing Unit (egy logikai mag) teljesítménye egy Intel Xeon 2,67 GHz vagy AMD Opteron 2,4 GHz számítási kapacitásának a negyedével ekvivalens teljesítményre képes [27]. Bár ez a mérési érték szolgáltatók között nem hordozható (a referencia-processzorok különbözősége és a mérési módszerek esetleges eltérősége miatt), adott szolgáltatók gépeit

viszont valóban jól leírják. Így tehát ha az egy számítási egységet tartalmazó gépen elvégezzük a mérést, akkor egy jól párhuzamosítható feladat esetén a mért értéket az adott gépekhez tartozó logikai egységek számával skálázhatjuk. Ez nem ront a mérések hitelességén, továbbá megkönnyíti adott szolgáltató értékelését és ezzel új szolgáltatók bevételét a modellbe, valamint lényeges költségmegtakarításokkal is járhat, hiszen nem kell minden géptípust példányosítani a mérés kedvéért (sok esetben az egy logikai magos gépek még akár ingyenesen is kipróbálhatóak).

A fentiek alapján mind a négy szolgáltató legkisebb, saját maggal rendelkező gépére elvégeztem a méréseket. Ezeket az eredményeket tartalmazza a 1. táblázat:

	Azure Small	**Amazon m1.small**	**HP Extra Small**	**Rackspace Small**
Whetstone (s)	229	252	155	241
Linpack (s)	319,563	438,379	194,656	380
DBench egészrendezés (s)	1,453	4,134	1,404	1,466
DBench string rendezés (s)	0,765	1,759	0,716	0,875
Dbench Huffmann-kódolás (s)	2,613	3,148	0,734	2,21
DBench prímkeresés (s)	75,36	53,368	13,245	76,39
DBench hozzárendelés (s)	41,25	26,764	21,823	20,105
Dhrystone (MIPS)	8447,52	4814,4	15243,44	8362,2
Livermore (MFLOP)	912,59	504,8	1478,2	887,61

1. táblázat - A szolgáltatók legkisebb gépeinek teljesítményeredményei

Mielőtt a modell felépítésének további részei is bemutatásra kerülnének, érdemes megvizsgálni a táblázatot. Látható, hogy a „legerősebb” gépei a legtöbb mérési eredmény szerint a HP-nak vannak. Különösen érdekes ez a megfigyelés annak fényében, hogy a Gartner elemzéseiben a HP a niche játékosok kategóriájában szerepel. Az Azure és a Rackspace gépei a legtöbb módszer alapján körülbelül egyenlő teljesítménnyel rendelkeznek (emlékeztetőül a Gartner elemzésében is azonos kategóriába estek).

Érdekes még, hogy az Amazon gépei bizonyos mérések szerint jobbak, mások szerint rosszabbak az előző kettőnél – ezért is jó, hogy több mérési keretrendszer eredményét aggregálva adja meg a modell a gépek teljesítményét.

Ezután – a 2. feltételezés alapján – a mérési eredményeket felskáláztam az adott gépekben lévő számítási egységeknek megfelelően minden szolgáltató gépire, majd a [0;1] tartományra képeztem az így kapott eredményeket úgy, hogy minden esetben a nagyobb értékek jelentsék a nagyobb teljesítményt, végül pedig aggregáltam az értékeket. A virtuális gépeket leíró teljes modellt a C függelék tartalmazza.

4.3.2 A tárhely modellje

A tárhely modelljét az 2. táblázat mutatja (a táblázat „Teljes használat" oszlopa intervallumokat tartalmaz; az intervallum alsó határa benne van az intervallumban, a felső nincs).

Teljes használat (TB-TB)	Azure ár ($/GB)	Amazon ár ($/GB)	Rackspace ár ($/GB)	HP ár ($/GB)[2]
0-1	0,095	0,095	0,1	0,1
1-50	0,08	0,08	0,09	0,1
50-200	0,07	0,07	0,085	0,1
200-500	0,07	0,07	0,08	0,1
500-1000	0,065	0,065	0,075	0,1
1000-1024	0,06	0,06	0,075	0,1

2. táblázat - Modell a tárhely költségének megállapítására

Ebből látható, hogy mennyi a tárolás fajlagos költsége.

Fontos, hogy bár egyszerű mérőszámról van szó, az alkalmazások tárhely-igényének megbecslése korántsem egyszerű feladat. A legtöbb szolgáltató adott napon a maximálisan elhasznált mennyiséget számlázza ki a felhasználóknak az adott napra, ezeket összeadja, majd elosztja 30-cal, más szolgáltatók viszont különböző időpontokban

[2] Mivel a HP tárhely-szolgáltatásai a dolgozat írásának pillanatában még bétában volt, így promóciós ajánlat keretében további 50% kedvezmény járt az árakból.

végeznek méréseket és ennek az eredményét (vagy éppen ezek átlagát) számlázzák adott napra.

4.3.3 A hálózati erőforrások modellje

Az általános hálózati forgalom és a CDN hálózati forgalom modelljét az alábbi táblázatok mutatják. Mivel az általános hálózati forgalom a felhőbe befelé minden szolgáltatónál ingyenes, a CDN pedig jellegéből adódóan mind letöltés, tehát szintén kifelé irányuló adatforgalom, így a táblázatok mindkét esetben a felhőből kifelé irányuló hálózati forgalomra vonatkoznak (a táblázatok „Teljes használat" oszlopai intervallumokat tartalmaznak; az intervallumok alsó határa benne van az intervallumokban, a felső nincs).

Teljes használat (TB-TB)	Azure ár ($/GB)	Amazon ár ($/GB)	Rackspace ár ($/GB)	HP ár ($/GB)
0-1	0	0	0,12	0
1-5	0	0,12	0,12	0,12
5-10	0,12	0,12	0,12	0,12
10-50	0,09	0,09	0,12	0,09
50-150	0,07	0,07	0,12	0,07
150-500	0,05	0,05	0,12	0,05

3. táblázat - Modell az általános hálózati forgalom költségeinek megállapítására

Teljes használat (TB-TB)	Azure ár ($/GB)	Amazon ár ($/GB)	Rackspace ár ($/GB)	HP ár ($/GB)
0-10	0,12	0,12	0,12	0,16
10-50	0,08	0,08	0,1	0,11
50-150	0,06	0,06	0,07	0,09
150-200	0,04	0,04	0,07	0,07
200-250	0,04	0,04	0,05	0,07

4. táblázat - Modell a CDN költségeinek megállapítására

A táblázatokból látható a kifelé menő adatok mennyiségének fajlagos költsége.

5 Költségoptimalizálás a modell alapján

Az előző fejezetben bemutatott modell lehetőséget ad arra, hogy egy alkalmazás költségeit szolgáltató-független módon leírjuk, majd szintén a modell alapján meghatározzuk a lehető legkisebb költséget, amivel az igények még kielégíthetőek.

Ezeknek az igényeknek a feltérképezéséhez a felhőalkalmazásokról különböző metrikák mentén adatokat kell gyűjteni, majd az így kapott eredményeket egyrészt a modellhez kell igazítani, másrészt a köztük lévő összefüggéseket is fel kell deríteni.

Ha az igények adottak, akkor a fejezet végén bemutatott két algoritmus a modellt felhasználva meg tudja határozni a költség szempontjából optimális és az igényeknek megfelelő konfigurációt.

5.1 Adatok gyűjtése

Az adatgyűjtés első lépéseként meghatároztam, hogy milyen paramétereit kell vizsgálni egy adott felhőalkalmazásnak. Ezek egy része általános: az összes, gépen futó folyamat processzor- és memória-felhasználására illetve I/O műveletek által írt/olvasott bájtokra vonatkoznak. A paraméterek másik része pedig a webes alkalmazásokra specifikus értékeket vizsgál: a beérkező HTTP-kérések száma illetve a webszerver folyamata által felhasznált memória és processzor.

Ezeknek az adatoknak a gyűjtésére minden szolgáltató ajánl valamilyen technológiai megoldást vagy szolgáltatást – ebben a fejezetben az Azure rendszerén keresztül mutatom be az adatgyűjtési lehetőségeket.

Az Azure elsősorban Windows operációs rendszereket futtató virtuális gépeket ajánl, ezekben pedig webes alkalmazásokat az Internet Information Services-ben (IIS) tudunk futtatni. Bár ezt IaaS szinten is felkonfigurálhatjuk, a PaaS megoldás alkalmazásával a mérési keretrendszer felállítása jóval egyszerűbbé tehető, és mivel a PaaS megoldás az IaaS megoldásra épít (lásd 2. ábra), így az eredmények hitelesek maradnak.

Az adatgyűjtést a Windows operációs rendszerekben teljesítmény-számlálók segítségével tudjuk elvégezni. A vizsgálni kívánt teljesítmény-számlálókat az Azure PaaS alkalmazások esetében egy XML alapú konfigurációs állományban kell felvenni Ez a

telepítőcsomag része; az alkalmazások telepítésekor az Azure saját operációs rendszere lefoglalja és konfigurálja az IaaS virtuális gépeket az esetleg szükséges szoftverekkel és az XML konfigurációs állományban felsorolt teljesítményszámlálókat is beállítja. Ezek folyamatosan futnak és a mérési eredményeket az Azure tárhely szolgáltatásába naplózzák.

Az adatgyűjtési módszerekkel kapcsolatban fontos követelmény, hogy adott szolgáltató esetén újrahasznosítható legyen. Az Azure esetében az XML konfigurációs állomány ennek a követelménynek eleget tesz (ennek teljes tartalmát lásd a D függelékben).

5.2 Adatok transzformálása és illesztése a modellhez

A gyűjtött adatokat a modellhez kell illeszteni. Ez elsősorban a processzor, másodsorban a memória felhasználásnak adataira vonatkozik. Ennek a módja a következő:

- Meghatároztam olyan mértékeket, amik függő mértékek; ez jelen esetben a processzor-kihasználtság és a memória-felhasználás (mind az összes, gépen futó folyamatra, mint pedig csak a webszerver folyamatára vonatkozóan).
- Meghatároztam a független mértékeket, amik változása befolyásolja a függő mértékeket. Ezek a beérkező HTTP-kérések illetve az I/O műveletek során olvasott/írt bájtok száma.
- A mérések időbélyegei alapján összerendeltem a függő és a független méréseket.
- Megkerestem azokat az értékpárokat, amik még a telítetlen állapotban (nem 100%-os kihasználtságú) kerültek rögzítésre. Ez azért fontos, mert ha például a memória-felhasználás eléri a maximum memóriát, akkor az ezután beérkező HTTP-kérések által generált munka már nem fog megjelenni a mérésekben, hiszen továbbra is csak annyit látunk az értékekből, hogy a teljes memória ki van használva. Ugyanez igaz a többi függő mértékre is.

- Ezek után meghatároztam a független és függő értékekre leginkább illeszkedő függvényt a még telítetlen tartomány adatainak alapján – ez a skálázódási függvény.
- A skálázódási függvénybe ezután behelyettesíthetőek azok a független mérési eredmények is, amelyekhez tartozó függő mérési értékek már a telített tartományba estek.
- A skálázási algoritmus bemenetében az igények végül a telítetlen tartomány mérési eredményei és a skálázódási függvénynek a telített tartományba eső helyeken felvett értékei együtt adják.
- A processzor esetében a mérési eredményt (processzor-kihasználtság százalékban) át kell alakítani a modellbeli processzorteljesítmény-mérőszámra. Ha a mérést egy olyan processzoron végeztük, amelynek a pontszáma *T* és a mérési eredmény *h* százalékos kihasználtságot mutat, akkor ez a mérési eredmény *T*h/100* értékké alakul.

5.3 Költségoptimalizálási algoritmusok

Miután a gyűjtött adatokat áttranszformáltam és a modellhez igazítottam, kidolgoztam két algoritmust, amelyek a modellt felhasználva ki tudják elégíteni a gyűjtött adatok által meghatározott igényeket. Az első megoldás naiv megközelítést használ, aminek az értékeléséből kiderül, hogy bizonyos eseteknek a kiértékelését nem tudja belátható időn belül elvégezni. Emiatt egy másik megoldás is kidolgozásra kerül, ami a lineáris programozás (illetve annak egy speciális esete, az egészértékű programozás) feladatára épít.

5.3.1 Naiv megoldás

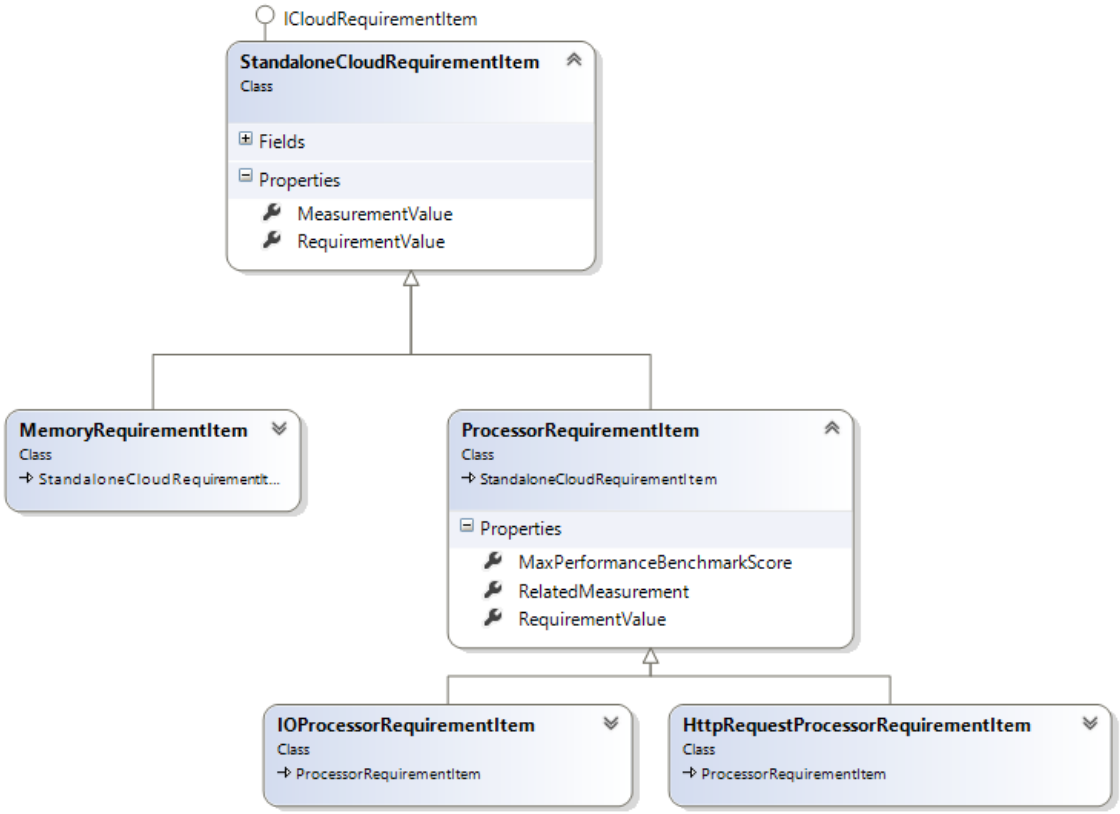
Az első megoldás az egyszerű, „minden lehetőség végignézése” (brute force) módszert követi. Az algoritmus ismertetése előtt fontos néhány fogalmat definiálni:

- igény: Adott egység, ami adott pillanatban egy virtuális gép valamely gyűjtött adatát tartalmazza. Ez lényegében a gyűjtött adatok modellhez illesztett formája. Fontos, hogy az igények csak olyan tulajdonságokat tartalmaznak, amik a gépek különböző megválasztásával befolyásolják a költséget; egy alkalmazás tárhely felhasználása vagy hálózati forgalma

által generált költség nem hangolható azáltal, hogy más konfiguráción futtatjuk, hiszen ezek nem virtuális gépektől függően kerülnek számlázásra. Ez a 4.1 fejezetben meghatározottaknak megfelelően azt jelenti, hogy processzor-kihasználtságra és memória-felhasználásra vonatkoznak ezek az igények.

- egyke igény: Olyan igény, amelynek a kielégítéséhez a felhasználók mindenképpen egy saját virtuális gépet szeretnének.
- összevonható igények: Olyan igények, amelyeket össze lehet vonni, hogy egy gép elégítse ki ezeket. Ezek együtt egy igénycsoportot alkotnak.

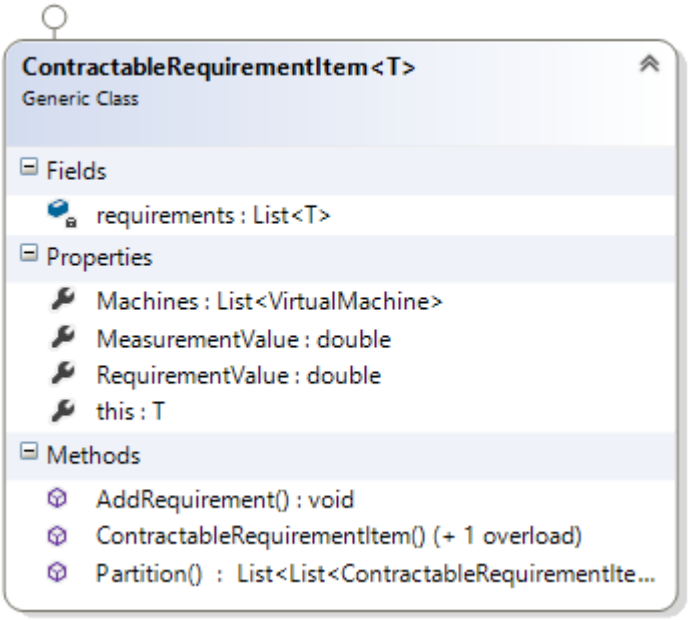
Az igények leírására szolgáló, az optimalizáló algoritmusokban felhasznált adatmodell elemeit a következő ábrák tartalmazzák.



5. ábra - Az igénytípusok adatmodellje

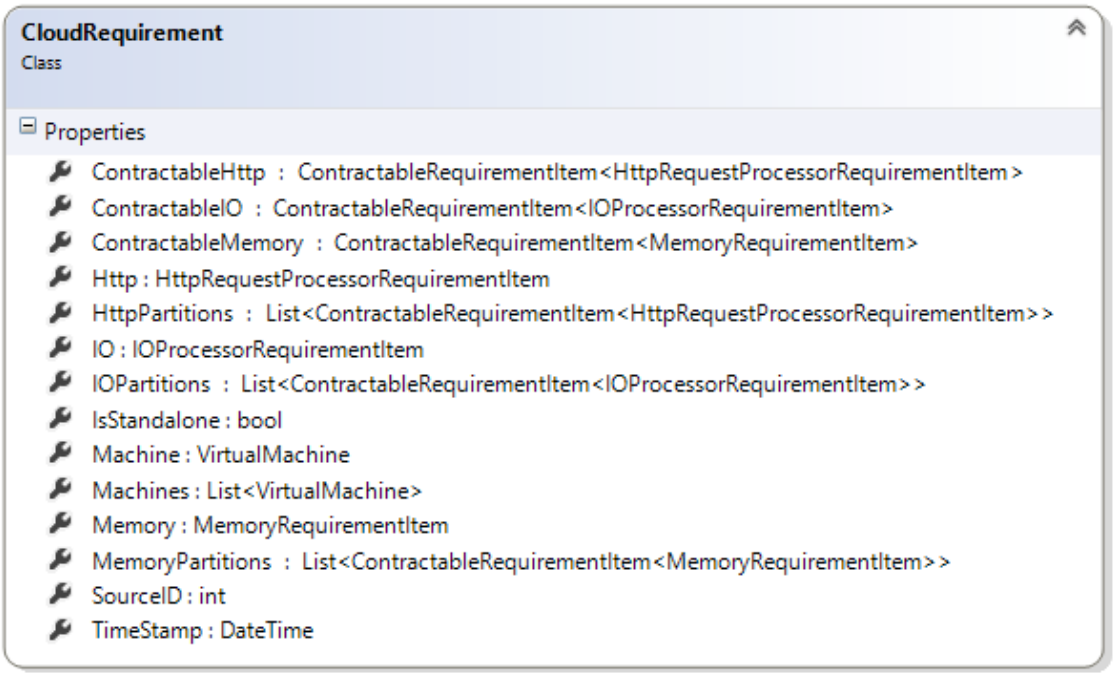
Az 5. ábra adatmodellje az igénytípusokat reprezentálja. Az előző fejezeteknek megfelelően memória- és processzor-kihasználtságra vonatkozó mérésekből származtatott igények leírására alkalmas ez a modell. Az egyke igénytípusok ennek az adatmodellnek közvetlenül megfeleltethetőek, az összevonható igények reprezentálására

bevezettem egy további típust, ami egy adott igénytípusból tartalmazhat többet egy listában:



6. ábra - Az összevonható igények reprezentálására szolgáló típus

Maga az igény az 5. ábra egyik típusának egy példányát vagy a 6. ábra összevonható igényeket reprezentáló típusának egy példányát tartalmazza néhány kiegészítő információval:



7. ábra - Egy igény reprezentálására szolgáló típus

A feladat tehát az igények kielégítése az adatgyűjtési és transzformációs lépés után. Elsőként az egyke igények kielégítését végzi el az algoritmus; ehhez végig kell nézni az összes, a modellben lévő virtuális gépet, majd kiválasztani azokat, amelyek ki tudják elégíteni az adott igényt, és végül közül kiválasztani a legolcsóbbat. Ennek megvalósítását az alábbi C# kódrészlet mutatja:

```
//az egyke igényeket a „standalon” változó tartalmazza
foreach (var s in standalone)
{
  Machines.ForEach(m => m.IsAdequateForProblem = true);
  //ha az igény memóriaára vonatkozik…
  if (s.Memory != null)
  {
    //kiszűröm az összes gépet, ami ezt a memória-igényt nem tudja kielégíteni
    //megadható egy határérték (ThresholdPercentage), ami fölé a metrika értéke
    //nem mehet a gépen, ezt figyelembe veszi az algoritmus
    Machines.Where(m =>
            m.MemoryinGB * MemoryTresholdPercentage < s.Memory.RequirementValue)
            .ForEach(item => item.IsAdequateForProblem = false);
  }
  //a memóriához hasonlóan a HTTP-kérések számával felskálázott processzor-
  //kihasználtságot vizsgálja az algoritmus
  if (s.Http != null)
  {
    Machines.Where(m =>
           m.ProcessorBenchmarkScore * ProcessorPerformanceTresholdPercentage
                                     < s.Http.RequirementValue).
               ForEach(item=> item.IsAdequateForProblem = false);
  }
  //a memóriához hasonlóan az I/O bájtok számával felskálázott processzor-
  //kihasználtságot vizsgálja az algoritmus
  if (s.IO != null)
  {
     Machines.Where(m =>
            m.ProcessorBenchmarkScore * ProcessorPerformanceTresholdPercentage
                                    < s.IO.RequirementValue)
                .ForEach(item=>item.IsAdequateForProblem = false);
  }
  //a szűrés után megmaradt gépek közül megkeresem a legolcsóbbat
  var temp=Machines.Where(m=>m.IsAdequateForProblem);
  s.Machine=temp.Count()==0 ? null : temp.Aggregate((m1,m2)=>
                  m1.DollarPricePerHour<m2.DollarPricePerHour ? m1 : m2);
}
```

A következő lépés az összevonható igények megfelelő elosztása és hozzájuk gépek foglalása. Az igényekről csupán azt tudjuk, hogy összevonhatóak, az ideális összevonás meghatározása a feladat része. Ehhez elsőként meg kell határozni az összevonható igények halmazának összes lehetséges partícionálását (felosztását). Ezek után minden egyes partícionálás költségét meg kell határozni, majd ezek közül a legolcsóbbat kiválasztani.

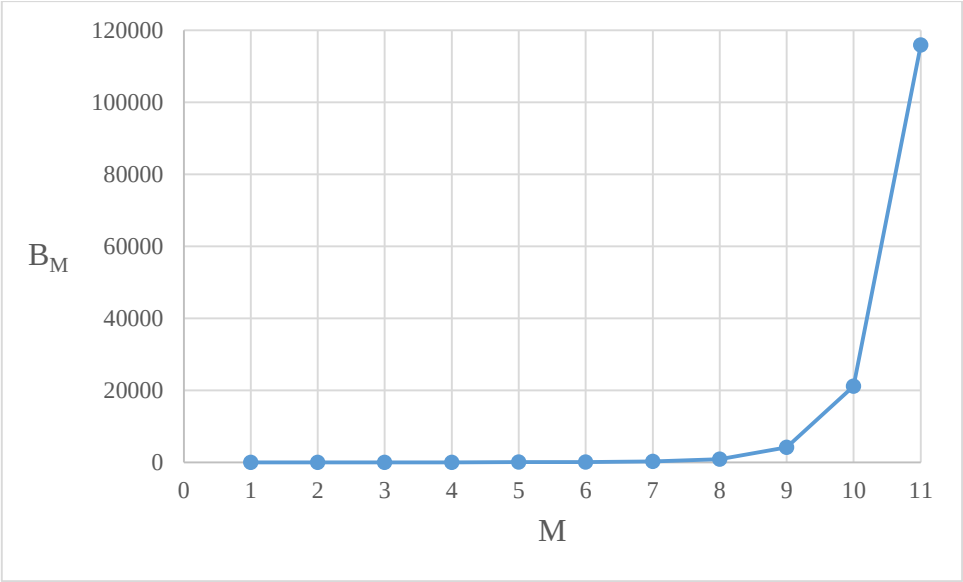
Egy adott partícionálás költségének meghatározása úgy történik, hogy az egyes partíciókban (az adott felosztásban szereplő halmazokban) lévő igényeket összeadjuk, így már önálló igényként lehet kezelni egy-egy partíciót; egy ilyen költségének a meghatározása (és hozzá megfelelő gép kiválasztása) pedig ugyanúgy történik, mint az egyke igények esetén. A teljes, adott partícionálás költsége pedig az egyes partíciók költségének az összege Az összevonható igények kiszámítására vonatkozó algoritmus implementációját az E függelék tartalmazza.

Bár az algoritmus egyszerű, meglehetősen nagy számításigényű. Adott *N* egyke, *M*, egy csoportba tartozó összevonható igény és *K* különböző virtuális gép esetén a számítási igény a következőképpen alakul:

- Az egyke igények meghatározásakor *N* igényhez kell *K* virtuális gépet végigpróbálni (a minimumot közben folyamatosan számon lehet tartani), így ennek lépésszáma *N***K*.
- A lépésszám másik fele az összevonható igények kielégítésének meghatározásának költsége. Ehhez első körben meg kell határozni az összes lehetséges partícionálását az *M* elemű halmaznak. Egy *M* elemszámú halmaz összes lehetséges partícionálásának számát a *M*. Bell-szám adja meg [28]. Az *M*. Bell-számot a következő képlet határozza meg:

$$B_M = \frac{1}{e} * \sum_{k=0}^{\infty} \frac{k^M}{k!}$$

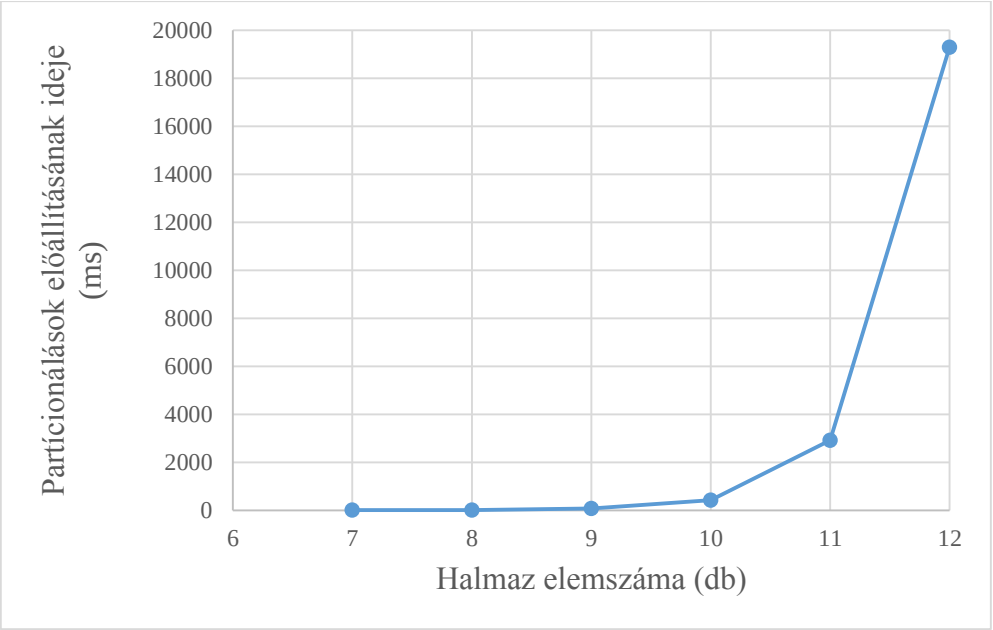
A képlet alapján az első néhány számot az 8. ábra mutatja:



8. ábra - Bell-számok

Látható, hogy a Bell-számok exponenciálisan nőnek, tehát már a lehetséges partícionálások felsorolása is exponenciális lépést jelent. Mivel minden egyes partícionálásra számításokat kell még elvégezni, ezért az algoritmusnak az összevonható igényeket feldolgozó része dominálja a teljes algoritmus (az egyke és az összevonható igények elosztását) lépésszámát.

Ebből a levezetésből látható, hogy az algoritmus lépésszáma miatt már egy, 10-11 összevonható igényt tartalmazó csoport esetén is kezelhetetlen, a gyakorlatban pedig még akár az is előfordulhat, hogy több összevonható igénycsoportot definiál a felhasználó, ezzel tovább bonyolítva a feladatot. Egy egyszerű laptopon elvégezve a méréseket a különböző elemszámú halmazok összes lehetséges partícionálásnak meghatározására, illetve annak időigényére a következő eredményeket kaptam (ezek a mérések összhangban vannak az előző ábrával, ahol szintén exponenciális növekedést lehet látni):



9. ábra - Halmazok összes lehetséges partícionálásának időigénye

A 9. ábra és a levezetés alapján ez az algoritmus már néhány összevonható igény esetén sem tudja elvárható időn belül megtalálni az ideális megoldást, így alternatívákat kell keresni.

5.3.2 Lineáris programozás

A lineáris programozás alapfeladata, hogy adott egyenlőtlenségrendszereket oldjunk meg úgy, hogy közben egy célfüggvényt minimalizálunk vagy maximalizálunk [29].

Az egyke igények meghatározása kezelhető az előző fejezetbeli levezetés alapján, így az összevonható igények kielégítésére kell egy hatékonyabb megoldás, ez lesz egy lineáris program, amit a következőképpen építettem fel a rendelkezésre álló modellelemek alapján (továbbra is K különböző típusú virtuális géppel dolgozunk):

- Minden virtuális géphez hozzárendeltem egy x_i változót; ennek az értéke azt fogja jelenteni, hogy az igény kielégítéséhez az i. típusú virtuális gépből az igények kielégítéséhez x_i darab példányt kell futtatni. A lehetséges virtuálisgép-típusokat a modell tartalmazza.
- Az i. virtuális gépnek egy adott paraméterének értéke p_i (ez lehet memória gigabájtban vagy a processzor pontszám; az egyszerűség kedvéért most csak a virtuális gépek egyetlen paraméterével mutatom be a lineáris

programot, ha memóriára és processzorra is vannak megkötések, akkor ennek megfelelően több paraméterértéket is fel kell venni). Ez a modellből kikereshető.

- Minden virtuális gépnek van egy k_i költsége, ez a modellből kikereshető.
- Az összevonható igények halmazának az összege b. Ha több különböző összevonható-igény halmaz is van, akkor ezek legyenek b_s
- Ezek után az egyenlőtlenség-rendszer, ami előáll:

$$\sum_{i=1}^{K} x_i * p_i \leq b_s$$

- A minimalizálandó költségfüggvény pedig:

$$\sum_{i=1}^{K} x_i * k_i + C$$

Ebben az egyenletben C értéke a felhasznált tárterület és a hálózati forgalom által generált költség. A következőkben látni fogjuk, hogy a lineáris programot meg lehet fogalmazni úgy, hogy mindig csak egyetlen szolgáltató ajánlatait vizsgálja. Ez fontos gyakorlati igény, hiszen üzemeltetési szempontból nem szerencsés, ha az alkalmazásaink egyes részei különböző felhőszolgáltatóknál futnak – egy szolgáltató esetén viszont C értéke konstansnak tekinthető az igény definíciója szerint, így az optimalizálás során nem kell vele számolni (a végső költségekhez viszont hozzájárul természetesen).

Az így előálló lineáris programot kell megoldani úgy, hogy x_i csak nemnegatív egész értékeket vehet fel (ettől lesz a feladata a lineáris programozás speciális esete, egészértékű programozás).

Ennek a lineáris programozási megoldásnak számos előnye van. Legfontosabb, hogy az előző megoldáshoz képest ez még nagyszámú összevonható igény esetén is elvárható időn belül ad megoldást. Szintén fontos, hogy lineáris programok megoldására vannak már jól bevált, általánosan használt szoftverek (például a Microsoft Excel táblázatkezelő, a LinGO szoftver vagy a Solver Foundation .NET környezetben), így az implementációval nem kell külön foglalkozni, csak az automatizációt kell megvalósítani.

További előny, hogy a modellbe különböző megkötéseket vehetünk be nagyon egyszerűen:

- Ha bizonyos géptípusokat (vagy akár egész szolgáltatókat) ki szeretnénk zárni a vizsgálatból, akkor az azoknak megfelelő x_i változókkal hozzá kell venni az $x_i=0$ egyenletet a modellhez. Így elégíthető ki az a gyakorlati igény, hogy az alkalmazásaink teljes egészében egy szolgáltatónál fussanak – ekkor az algoritmus többszöri futtatásával meghatározhatjuk, hogy az egyes szolgáltatóknál milyen költségű konfigurációval elégíthetőek ki az igények, majd ezeket összehasonlítva egyszerűen kiválasztható a legoptimálisabb.
- Ha azt szeretnénk, hogy az igények kielégítését legalább t darab géppel tegyük meg (ez terheléselosztási vagy rendelkezésre állási szempontok miatt lehet szükséges), akkor a következő egyenlőtlenséget kell hozzávenni a lineáris programhoz:

$$\sum_{i=1}^{K} x_i \geq t$$

- Hasonlóan megszabhatjuk azt is, hogy legfeljebb r gépet szeretnénk felhasználni:

$$\sum_{i=1}^{K} x_i \leq r$$

6 Ipari felhasználás és fejlesztési lehetőségek

Zárásként a következő fejezetek az algoritmusok konkrét rendszerekben történő felhasználási lehetőségeit és ezek implementálási kérdéseit mutatják be röviden.

6.1 Automatikus skálázás

A dolgozatban bemutatott módszer megadja a felhasználóknak, hogy melyik szolgáltatónál milyen konfigurációt érdemes lefoglalniuk a felmerülő igények költséghatékony kielégítéséhez. Jelen fázisban a cél csupán ennyi volt – kidolgozni egy skálázási módszert, ami komplex metrikák alapján képes döntést hozni.

Most már tehát a felhasználók az algoritmust lefuttatva és az eredményeket megvizsgálva a szolgáltatók által biztosított felületen le tudják foglalni a megfelelő erőforrásokat (esetleg a szabályalapú skálázó megoldások hatékonyabb felhasználásával használhatják). Mivel azonban a legtöbb szolgáltató biztosít valamilyen programozási felületet, amivel a webes felület funkcióit (és gyakran annál még többet) megfelelő utasításhívásokkal is el lehet végezni, így az optimalizációt végző algoritmus is lefoglalhatná vagy felszabadíthatná a megfelelő erőforrásokat. Ez az itt bemutatott módszer elsődleges ipari felhasználási lehetősége.

6.2 Fejlesztési tervek

A fejlesztési tervek mind azt a célt szolgálják, hogy az előző fejezetben röviden bemutatott automatikus skálázó elkészülhessen. Ehhez legtöbb esetben a jelenlegi módszer egyes lépéseinek automatizálásra van még szükség:

- Szükség van az elterjedtebb programozási platformokhoz kidolgozni egy olyan módszert, amivel a már meglévő felhőalkalmazásokról az algoritmus számára szükséges adatok könnyen összegyűjthetők. Ez a már bemutatott prototípusban az Azure esetében egyszerűen a megadott teljesítménymutatók felvétele a telepítőcsomag konfigurációs állományába. Más platformok (felhőplatformok vagy szoftverplatformok) esetén hasonló megoldásokat kell kidolgozni. Ezek lehetnek hasonló konfigurációs állományok vagy akár egész szoftvermodulok.
- A gyűjtött adatok lehívását is automatizálni kell. Ehhez szükséges egy megfelelő interfész bevezetése és implementálása. A legtöbb szolgáltató ad lehetőséget arra,

hogy a gyűjtött adatokat lekérdezhessük (az Azure például egy OData interfészen keresztül biztosít hozzáférést a teljesítményszámlálók adataihoz). Ezek után az adatok transzformációját, modellhez igazítását is automatizálni kell.

- Magának a modellnek a felépítését és karbantartását is emberi beavatkozás nélkül kell megoldani. Mivel azonban a szolgáltatók nem ajánlanak semmilyen API-t az éppen aktuális árazás lekérdezésére, ezért ezt a szolgáltatók weblapjainak feldolgozásával, adatbányászati módszerekkel kell előállítani.
- Végül pedig meg kell valósítani az előző fejezetben leírt, skálázó utasításokat végrehajtó modult (természetesen ezt is egy megfelelő interfészen keresztül, így támogatható több szolgáltató is transzparens módon).
- A kutatás egy másik vonalán pedig minél több felhőszolgáltatót érdemes bevonni a modellbe.
- A modell koncepciója is bővíthető még; a legtöbb szolgáltató tranzakcionális költségeket is számol (pl.: írások vagy olvasások száma, http kérések száma alapján); bár ezek a költségek egy kisebb alkalmazás esetén elenyészőek (sőt valamennyi tranzakció gyakran ingyenes is), egy nagyobb alkalmazás esetén valós költséget jelentenek. Ezek számlázása viszont nagyon eltérő szolgáltatónként, így ennek bevezetése önmagában is egy komplex feladat.

7 Összefoglalás

A dolgozat célja egy felhőalkalmazások költség-optimalizálását megvalósító, szolgáltató-független rendszer kidolgozása volt. Az előző fejezetek ennek lépéseit mutatták be.

A munka pontos célja és előrevetítése után a dolgozat a számítási felhő jelentőségét és technikai aspektusait mutatta be, kitérve a számítási felhő születésére, definíciójára és elterjedtségére. A technikai fejezetek a szolgáltatás-modellként értelmezett felhő konkrét megvalósítási lehetőségeit tárgyalták, megvizsgálva olyan kérdéseket, hogy az egyes lehetőségek kiknek biztosítanak hozzáférést a virtualizált erőforrásokhoz vagy milyen erőforrások kerülnek virtualizálásra.

Miután röviden kitértem a számítógépek teljesítmények mérésével kapcsolatos problémákra, rátértem a szolgáltató-független költség-optimalizálás bemutatására. Ehhez kidolgoztam egy modellt, amiben különböző szolgáltatók ajánlatai egységes, összehasonlítható módon leírhatóak. A modell megalkotásának első lépése az volt, hogy az előző fejezetekben bemutatott technikai aspektusok alapján kiválasztottam azokat a szolgáltatástípusokat, amelyek vizsgálata jól általánosítható eredményeket ad és egy tipikus felhőalkalmazás költségeit meghatározzák. Az így meghatározott szolgáltatások adják azokat a tulajdonságokat, amelyek alapján a modell a szolgáltatókat összehasonlíthatóvá teszi. A modellbe négy szolgáltató, az Amazon, a Microsoft, a Rackspace és a HP ajánlatai kerültek felvételre.

A modell megalkotása után meghatároztam olyan metrikákat, amelyek mentén egy tipikus felhőalkalmazást monitorozni érdemes ahhoz, hogy az alkalmazás erőforrás-felhasználását jól le tudjuk írni. Bemutattam a módszereket, amelyekkel a gyűjtött adatok a modellekhez igazíthatóak és a köztük lévő összefüggések leírhatóak. Ezeknek segítségével az adatokból előállnak azok az igények, amelyek a költség-optimalizálás bemeneteként szolgálnak. Az adatgyűjtési feladatok úgy lettek definiálva, hogy minden felhőszolgáltató ajánljon lehetőséget a megvalósításukra; a dolgozat a Microsoft felhőszolgáltatásán keresztül mutatja be a konkrét módszert.

A dolgozat bemutatott két költség-optimalizáló algoritmust; az egyik egy naiv, minden lehetőséget megvizsgáló („brute force”) megoldás, a másik pedig egy lineáris

programozáson alapuló módszer. Az algoritmusok leírása során a két algoritmust értékeltem teljesítmény és testreszabhatóság szempontjából.

A dolgozat záró fejezete röviden kitért arra, hogy a költség-optimalizálási algoritmusokat hogyan lehet felhasználni automatikus, intelligens skálázást megvalósító rendszerekben és azokra a feladatokra, amelyek megvalósítása ehhez még szükséges.

Irodalomjegyzék

[1] J. Jones, Empires of Light: Edison, Tesla, Westinghouse, and the Race to Electrify the World, New York: Random House Inc. Publishing, 2004.

[2] About AWS, [Online]. Available: http://aws.amazon.com/about-aws/. [Hozzáférés dátuma: 2013. október 18.].

[3] Rackspace, „Rackspace Hosting Reports Nine in Ten IT Decision Makers in National Survey Have Positive View of Cloud Computing," 25 június 2012. [Online]. Available: http://www.rackspace.com/blog/newsarticles/nine-in-ten-it-decision-makers-in-national-survey-have-positive-view-of-cloud-computing/110/. [Hozzáférés dátuma: 2013. október 18.].

[4] S. Lauchlan, „Gartner: IaaS takes over from traditional data centre outsourcing," 2012. augusztus 20. [Online]. Available: http://www.businesscloud9.com/content/gartner-iaas-takes-over-traditional-data-centre-outsourcing/11472. [Hozzáférés dátuma: 2013. október 18.].

[5] S. Kar, „Gartner: Cloud Spending on Public Services is Growing Rapidly," 2012. július 23. [Online]. Available: http://cloudtimes.org/2012/07/23/gartner-spending-public-services/. [Hozzáférés dátuma: 18 október 2013].

[6] J. McKendrick, „Majority of Companies Expanding Cloud Computing Skills: Survey," 2012. július 24. [Online]. Available: http://www.forbes.com/sites/joemckendrick/2012/07/24/majority-of-companies-expanding-cloud-computing-skills-survey/. [Hozzáférés dátuma: 2013. október 18.].

[7] T. Barnett, „Updated Global Cloud Index: Revised Forecast Shows Clear Signs of Continued Data Center Virtualization,” 2012. október 23. [Online]. Available: http://blogs.cisco.com/news/updated-global-cloud-index-revised-forecast-shows-clear-signs-of-continued-data-center-virtualization/110. [Hozzáférés dátuma: 2013. október 18.].

[8] X. Cruz, „Cloud Services Expected to Reach $100 Billion,” 2012. szeptember 16. [Online]. Available: http://cloudtimes.org/2012/09/16/cloud-services-100-billion/. [Hozzáférés dátuma: 2013. október 18.].

[9] A. Higgins, „What's Eating Software-Services Companies?,” 2013. március 29. [Online]. Available: http://beta.fool.com/higginsaustin/2013/03/29/what-is-eating-at-software-as-a-service-companies/27937/. [Hozzáférés dátuma: 2013. október 18.].

[10] Gartner, „Gartner's Hype Cycle for Emerging Technologies,” Gartner, New York, 2012.

[11] P. Mell és T. Grance, „The NIST definition of cloud computing (draft),” *NIST special publication,* 800. kötet, 145. szám, 7. oldal, 2011.

[12] L. Carlson, Programming for PaaS, Sebastopol: O'Reilly, 2013.

[13] B. Farkas, G. Kovács, I. Király, A. Turóczy, T. Kőnig, A. Érsek, M. Safranka, D. Fülöp, K. Pellek és B. Kiss, Windows Azure lépésről lépésre, Budapest: Jedlik Oktatási Stúdió, 2013.

[14] The Autoscaling Application Block, [Online]. Available: http://msdn.microsoft.com/en-us/library/hh680892(v=pandp.50).aspx. [Hozzáférés dátuma: 2013. október 18.].

[15] Amazon Auto Scaling, [Online]. Available: http://aws.amazon.com/autoscaling/. [Hozzáférés dátuma: 2013. október 18.].

[16] D. Black és J. Thomas, „How Gartner Evaluates Vendors and Markets in Magic Quadrants and MarketScopes,” Gartner, New York, 2013.

[17] L. Leong, D. Toombs, B. Gill, G. Petri és T. Haynes, „Magic Quadrant for Cloud Infrastructure as a Service," Gartner, New York, 2013.

[18] T. Smith, „Megahertz myth," *The Guardian*, 2002. február 27.

[19] J. Napper és P. Bientinesi, „Can cloud computing reach the top500?," in *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, 2009.

[20] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman és N. Wright, „Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010.

[21] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer és D. Epema, „A performance analysis of EC2 cloud computing services for scientific computing," in *Cloud Computing*, Springer, 2010, 115-131 oldal.

[22] H. J. Curnow és B. A. Wichmann, „A synthetic benchmark," *The Computer Journal*, 19. kötet, 1. szám, 43-49 oldal, 1976.

[23] A. R. Weiss, „Dhrystone benchmark: History, analysis, scores and recommendations," *White paper, EEMBC Certification Laboratories, LLC*, 2002.

[24] J. J. Dongarra, P. Luszczek és A. Petitet, „The LINPACK Benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, 15. kötet, 9. szám, 803-820 oldal, 2003.

[25] F. H. McMahon, „Llnl fortran kernels: Mflops," *Lawrence Livermore Laboratories, Livermore, CA*, 1984.

[26] Original NBench Documentation (archived snapshot), [Online]. Available: http://www.tux.org/~mayer/linux/byte/bdoc.pdf. [Hozzáférés dátuma: 2013. október 18.].

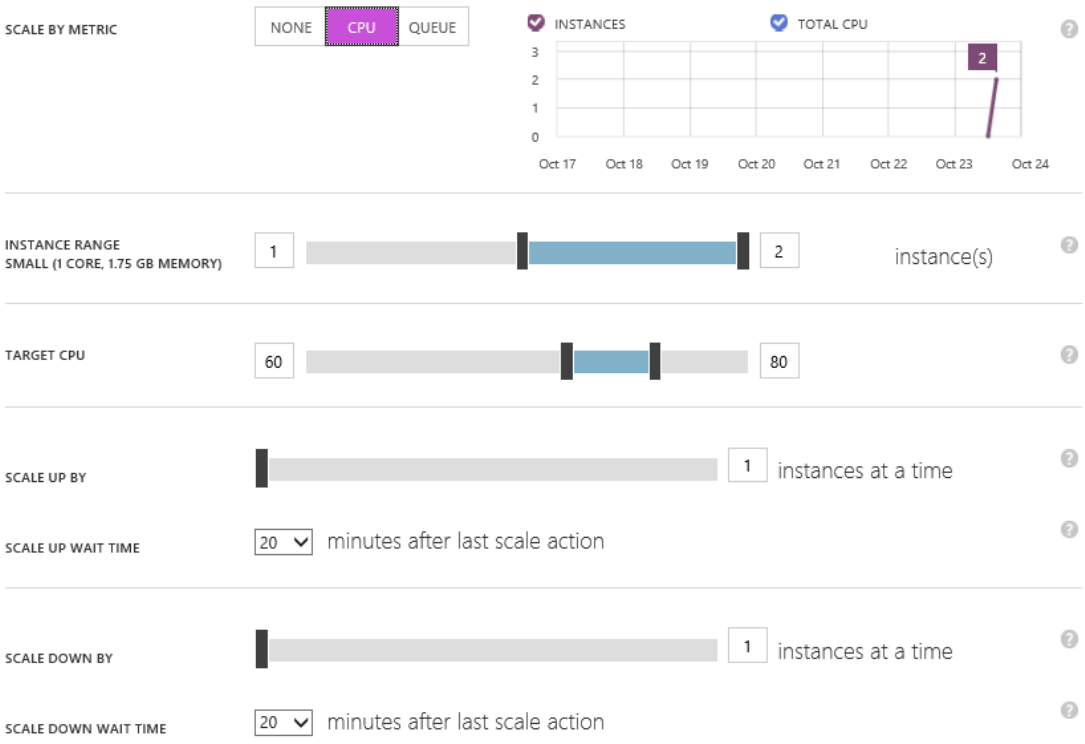
[27] HP Cloud Pricing, [Online]. Available: http://www.hpcloud.com/pricing. [Hozzáférés dátuma: 2013. október 18.].

[28] M. Gardner, „The Bells: versatile numbers that can count partitions of a set, primes and even rhymes,” *Scientific American*, 24-30 oldal, 1978.

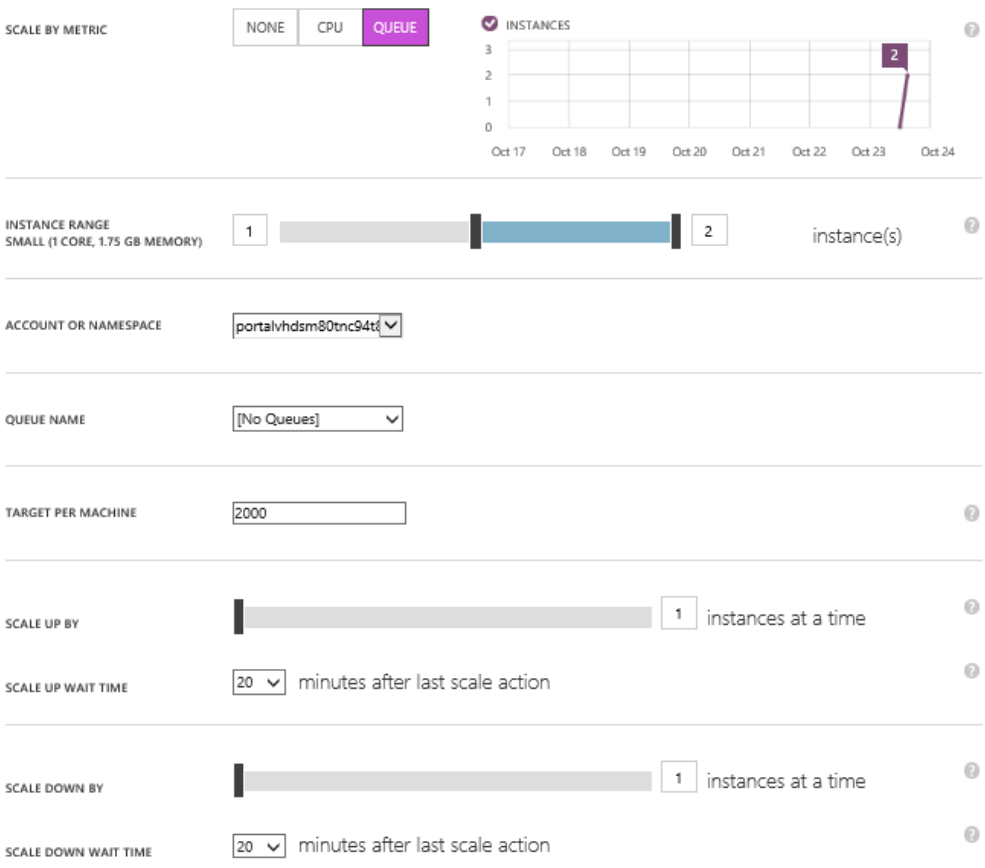
[29] T. Jordán, D. Szeszlér és A. Recski, Rendszeroptimalizálás, Budapest: TypeTex, 2010.

A függelék: Azure automatikus skálázás

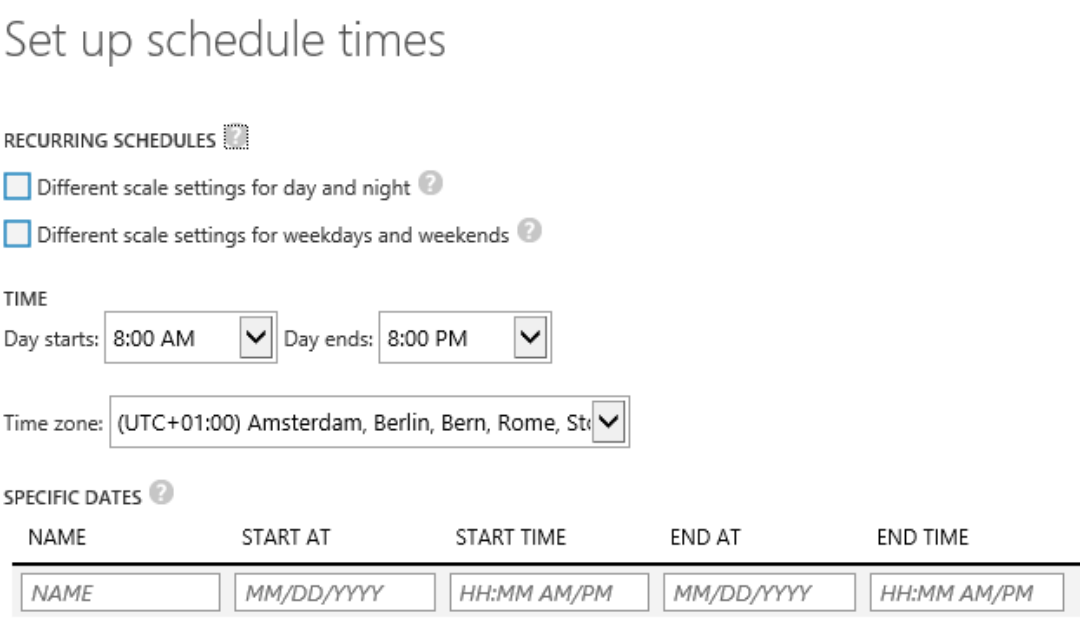
A következő néhány képernyőkép az Azure egyszerűbb skálázási megoldását mutatja be. A funkció a felhasználók számára további költségek nélkül igénybe vehetőek a webes felügyeleti portálon:



Skálázás Azureban CPU-kihasználtság alapján



Skálázás Azure-ban adott üzenetsorban lévő üzenet számai alapján



Szabályok megadása időintervallumokban Azure-ban

B függelék: Enterprise Library Autoscaling Application Block konfiguráció

Az Enterprise Library Autoscaling Application Block az Azure skálázási megoldásai közül a testre szabhatóbb és rugalmasabb. Ez is szabályalapú kiértékelést használ, viszont a szabályokban több metrikát definiálhatunk, valamint a szabályok között is meghatározhatunk hierarchikus kapcsolatokat.

Az alábbi egyszerű konfiguráció két megkötést tartalmaz; az egyik mindig aktív, a másik ez előbbi írja felül csúcsidőben; ezek a gépek minimális és maximális számát adják meg. Van továbbá két reaktív szabály is a modellben; az egyik növeli a gépek számát, ha az elmúlt 30 percben az átlagos CPU kihasználtság 85% fölött volt, a másik pedig csökkenti, ha 20% alatt volt.

```
<rules>
  <constraintRules>
    <rule name="Default" description="Always active"
          enabled="true" rank="1">
      <actions>
        <range min="2" max="5" target="RoleA"/>
      </actions>
    </rule>
    <rule name="Peak" description="Active at peak times"
          enabled="true" rank="100">
      <actions>
        <range min="4" max="6" target="RoleA"/>
      </actions>
      <timetable startTime="08:00:00" duration="06:00:00">
        <daily/>
      </timetable>
    </rule>
  </constraintRules>
  <reactiveRules>
    <rule name="ScaleUp" description="Increases instance count"
          enabled="true" rank="10">
      <when>
        <greater operand="Avg_CPU_RoleA" than="85"/>
      </when>
      <actions>
        <scale target="RoleA" by="1"/>
      </actions>
    </rule>
```

```
      <rule name="ScaleDown" description="Decreases instance count"
          enabled="true" rank="10"><when>
        <less operand="Avg_CPU_RoleA" than="20"/>
      </when>
      <actions>
        <scale target="RoleA" by="-1"/>
      </actions>
     </rule>
   </reactiveRules>
  <operands>
    <performanceCounter alias="Avg_CPU_RoleA"
        performanceCounterName="\Processor(_Total)\% Processor Time"
        aggregate="Average" source="RoleA" timespan="00:30:00"/>
  </operands>
</rules>
```

C függelék: A modell virtuális gépeket leíró része

Virtuális gép			Processzor teljesítmény pontszám	Memória (GB)	Ár ($/óra)
Azure	**Standard**	Small	1,7997	1,75	0,09
		Medium	3,9740	3,5	0,18
		Large	5,1166	7	0,36
		Extra Large	5,7989	14	0,72
	Memory	A6	5,1166	28	1,02
		A7	5,7989	56	2,04
Hp		Extra Small	3,8649	1	0,06
		Small	5,0458	2	0,12
		Medium	5,7310	4	0,24
		Large	5,9059	8	0,48
		Extra Large	6,9079	16	0,96
		Double Extra Large	7,9881	32	1,92
Rackspace		1G	1,6552	1	0,08
		2G	3,9006	2	0,16
		4G	4,8195	4	0,32
		8G	5,5372	8	0,58
		15G	5,8248	15	1,08
		30G	6,0049	30	1,56
Amazon	**General purpose**	m1.small	0,3030	1,7	0,091
		m1.medium	3,2013	3,75	0,182
		m1.large	4,6815	7,5	0,364
		m1.xlarge	5,4839	15	0,728
		m3.xlarge	5,8644	15	0,78
		m3.2xlarge	6,3555	30	1,56
	Memory optimized	m2.xlarge	5,2868	17,1	0,51
		m2.2xlarge	4,1812	34,2	1,02
		m2.4xlarge	6,3555	68,4	2,04
		cr1.8xlarge	7,7983	244	3,831
	Compute optimized	c1.medium	4,9900	1,7	0,225
		c1.xlarge	6,1646	7	0,9
		cc2.8xlarge	7,7983	60,5	2,97
	GPU	cg1.4xlarge	6,5607	22,5	2,6
	Storage optimized	hi1.4xlarge	6,5992	60,5	3,58
		hs1.8xlarge	6,5992	117	4,931

D függelék: Azure PaaS konfigurációs állomány

```
<DiagnosticMonitorConfiguration
     configrationChangePollInterval="PT1M" overallQuotaInMB="4096"
       xmlns="http://schemas.microsoft.com/ServiceHosting/2010/10/
             DiagnosticsConfiguration">
<!--A gyűjtött adatok 5 percenként kerülnek át a perzisztens tárba-->
  <PerformanceCounters scheduledTransferPeriod="PT5M">
    <!--A sampleRate értéke alapján 5 másodpercenként történik egy mérés-->
    <PerformanceCounterConfiguration
       counterSpecifier="\ASP.NET Applications(__Total__)\Requests Total"
       sampleRate="PT5S" />
    <PerformanceCounterConfiguration
      counterSpecifier="\ASP.NET Applications(__Total__)\Requests/Sec"
      sampleRate="PT5S" />
    <PerformanceCounterConfiguration
      counterSpecifier="\ASP.NET Applications(__Total__)\Request Execution Time"
      sampleRate="PT5S" />
    <PerformanceCounterConfiguration
       counterSpecifier="\Memory\Available Bytes"
       sampleRate="PT5S" />
    <PerformanceCounterConfiguration
       counterSpecifier="\Processor(_Total)\% Processor Time"
       sampleRate="PT5S" />
    <PerformanceCounterConfiguration
       counterSpecifier="\Process(w3wp)\% Processor Time"
       sampleRate="PT5S" />
    <PerformanceCounterConfiguration
       counterSpecifier="\Process(w3wp)\Private Bytes"
       sampleRate="PT5S" />
    <PerformanceCounterConfiguration
       counterSpecifier="\Process(w3wp)\IO Data Bytes/sec"
       sampleRate="PT5S" />
    <PerformanceCounterConfiguration
       counterSpecifier="\Process(w3wp)\IO Data Operations/sec"
       sampleRate="PT5S" />
  </PerformanceCounters>
</DiagnosticMonitorConfiguration>
```

E függelék: Összevonható igények hozzárendelése gépekhez

A függelék az összevonható igények virtuális gépekhez rendelésének implementációjának fontosabb részleteit mutatja be. .NET környezetben, C# nyelven (a hatékonyabb bemutatás kedvéért a változók inicializálását nem tartalmazza a részlet).

```
//a „contractable” változó tartalmazza az összevonható igényeket
foreach (var c in contractable)
{
  //ha az összevonható igény memóriára vonatkozik, akkor meghatározzuk az összes
  //partíciót és ezek számát egy memóriaigényeket tartalmazó változóba
  if (c.ContractableMemory!=null)
  {
    //a Partition() bővítőmetódus az összes partíciót meghatározza;
    //egy partíciót egy listában ad vissza, ezek listáját adja vissza a metódus
    //a metódus implementációja egyszerű rekurzív felsorolást használ
    memparts = c.ContractableMemory.Partition();
    partcount = memparts.Count;
  }
  //ha az összevonható igény I/O műveletekből származtatott processzor-
  //kihasználtságra vonatkozik, akkor meghatározzuk az összes
  //partíciót és ezek számát erre dedikált változókba
  if (c.ContractableIO != null)
  {
    ioparts = c.ContractableIO.Partition();
    partcount = memparts.Count;
  }
  //ha az összevonható igény http-kérésekből származtatott processzor-
  //kihasználtságra vonatkozik, akkor meghatározzuk az összes
  //partíciót és ezek számát erre dedikált változókba
  if (c.ContractableHttp != null)
  {
    httpparts = c.ContractableHttp.Partition();
    partcount = memparts.Count;
  }

  //megvizsgáljuk egyesével a lehetséges partícionálásokat
  for (int i = 0; i < partcount; i++)
  {

    //Az aktuálisan vizsgált partícionálást az igény típusától függően a
    //számosságával együtt eltesszük egy erre dedikált változóba
    if (memparts != null)
    {
      memsubparts = memparts[i];
      subpartcount = memsubparts.Count;
    }
    if (httpparts != null)
    {
      httpsubparts = httpparts[i];
      subpartcount = httpsubparts.Count;
    }
    if (ioparts != null)
```

```
    {
      iosubparts = ioparts[i];
      subpartcount = iosubparts.Count;
    }

    //az aktuális partícionálás minden egyes halmazán végigmegyünk
    //és egy megfelelő változóba eltesszük
    for (int j = 0; j < subpartcount; j++)
    {
      if (memsubparts != null)
      {
        currentmem = memsubparts[j];
      }
      if (iosubparts != null)
      {
         currentio = iosubparts[j];
      }
      if (httpsubparts != null)
      {
         currenthttp = httpsubparts[j];
      }

      //meghatározzuk a gépeket, amik ki tudják elégíteni az igényt
      //a Machines váltóz tartalmazza a modell virtuális gépekre vonatkozó részét
      Machines.ForEach(m => m.IsAdequateForProblem = true);


      //a gépek, amik nem felelnek meg az igényeknek, kikerülnek a listából
      //minden metrikához megadható egy határérték (threshold), ami fölé nem
      //mehet az adott metrika a gépen (pl.: a memóriakihasználtság nem mehet
      //90% fölé)
      //a Requirement tulajdonság tartalmazza a halmazban lévő igények összegét
      if (currentmem != null)
      {
         Machines.Where(m => m.MemoryinGB * MemoryTresholdPercentage
                                          < currentmem.RequirementValue).
                  ForEach(item => item.IsAdequateForProblem = false);
      if (currenthttp != null)
      {
         Machines.Where(m =>
              m.ProcessorBenchmarkScore * ProcessorPerformanceTresholdPercentage
                                          < currenthttp.RequirementValue)
                  .ForEach(item =>item.IsAdequateForProblem = false);
       }
       if (currentio != null)
       {
       Machines.Where(m =>
              m.ProcessorBenchmarkScore * ProcessorPerformanceTresholdPercentage
                                          < currentio.RequirementValue)
              .ForEach(item => item.IsAdequateForProblem = false);
       }

       //meghatározzuk azt a gépet, ami az adott partícionálás adott
       //halmaza által reprezentált igények összességét a legkisebb költséggel
       //tudja kielégíteni
       var temp = Machines.Where(m => m.IsAdequateForProblem);
       var machine = temp.Count()==0 ? null :
                    temp.Aggregate((m1, m2) =>
                    m1.DollarPricePerHour < m2.DollarPricePerHour ? m1 : m2);
       machines[i].Add(machine);
     }
   }
```

```
   //meghatározzuk melyik partícinoálás a legolcsóbb
   var min = double.MaxValue;
   var minpart = -1;
   for (int i = 0; i < machines.Count; i++)
   {
     //a partícionálás költsége az egyes partíciókhoz tartozók gépek költségének
     //összege
     var temp = machines[i].Sum(m => m.DollarPricePerHour);
     if (temp < min)
     {
        temp = min;
        minpart = i;
     }
   }
   //attól függően, hogy mire vonatkozott az igény, a megfelelő tulajdonságba
   //kerül beírásra a virtuális gép
   c.Machines = machines[minpart];
   if (ioparts != null)
   {
     c.IOPartitions = ioparts[minpart];
   }
   if (httpparts != null)
   {
     c.HttpPartitions = httpparts[minpart];
   }
   if (memparts != null)
   {
     c.MemoryPartitions = memparts[minpart];
   }
}
```