KÖLTSÉGHATÉKONYSÁG CLOUD ALAPÚ RENDSZEREKBEN - ERŐFORRÁSALLOKÁCIÓ PRIVÁT IAAS CLOUDOK ESETÉN

Hartung István, hartung@iit.bme.hu
Dr. Goldschmidt Balázs, balage@iit.bme.hu
Budapesti Műszaki és Gazdaságtudományi Egyetem, Irányítástechnika és Informatika Tanszék
1117 Budapest, Magyar tudósok krt. 2

Tematika

Egy kisebb méretű, vagy frissen alakult vállalat számára azért is lehet vonzó cloud rendszerek használata, mivel jelentős költségeket spórolhatnak meg azzal, hogy nem szükséges hardvert vásárolni, valamint azok konfigurációs és karbantartási költségeit megfizetni. Ugyanez igaz olyan cégek vagy hivatalok esetén, amelyek szolgáltatásait bizonyos időközönként jelentősen többen veszik igénybe, így az év jelentős részében kihasználatlan lenne az infrastruktúrájuk. Hasonló felesleges erőforrás allokációt okozhatnak bizonyos szoftverkonfigurációk, melyek elemei különböző operációs rendszert igényelnek, emiatt különálló fizikai számítógépre kell őket telepíteni, holott nem használják ki azok teljes kapacitását.

A legtöbb privát cloud szolgáltatás még gyerekcipőben jár az automatikus erőforrás-allokáció, valamint erőforrás-elosztás tekintetében, és ez különösen igaz a nyílt forráskódú IaaS keretrendszerek esetén, amelyekben többnyire csak alapvető algoritmusok kerültek implementálásra. Ezen algoritmusok nem rendelkeznek olyan képességekkel, melyek valódi energiafogyasztás-csökkenést eredményeznek. Kutatásunk a különböző nyílt forráskódú IaaS rendszerek virtuális gép migrációs és fizikai erőforrás allokációs lehetőségeinek vizsgálatára irányul, valamint ezek fejlesztését célozza meg, hogy lehetővé tegyük a nem használt hardver erőforrások energiatakarékos üzemmódba való átkapcsolását.

Bevezetés

A számítási felhő használatának számos oka lehet egy vállalatnál, de végső soron a költségmegtakarítás a fő célja bevezetésének.[1] A költségek csökkentését okozhatja, hogy cloud használatával hatékonyabbá válik a fejlesztés, a tesztelés, a bonyolult és hosszú telepítési ciklus lerövidül és hibamentessé válik. Szükségtelenné válik új szerverek vásárlása, konfigurálása és karbantartása, mivel gyorsabban és olcsóbban hozzá lehet jutni virtualizáció segítségével a szükséges gépekhez. A cég korábban meglevő infrastruktúráját sokkal hatékonyabban tudja kihasználni az által, hogy az erőforrásokat szétosztja, és a nem használt hardvert energiatakarékos állapotban tartja.

Jelen kutatás a erőforrások optimális szétosztását lehetővé tevő infrastruktúrát nyújtó felhő rendszerek vizsgálatát célozza meg. Célunk a különböző nyílt forráskódú rendszerek jelenlegi erőforrás allokációs algoritmusainak vizsgálata, összevetése, majd a különböző szempontok szerinti legjobb algoritmusok megtalálása, implementálása és tesztelése. A kutatáshoz megtörtént az általunk kiválasztott CloudStack cloud telepítése és konfigurációja egy több szerverből álló saját belső rendszeren.

A következő alfejezetek a cloud technológia alapjainak bemutatását célozzák meg, ezután olvasható a vizsgált három nyílt forráskódú rendszer architektúrájának bemutatása, majd a meglevő algoritmusok ismertetése, végül a kutatási irány kijelölése történik meg.

Cloud technológia

A számítási felhő alapú technológia infrastruktúrát (*Infrasturcture as a Service - IaaS*), platformot (*Platform as a Service - PaaS*) vagy szoftvert (*Software as a Service - SaaS*) nyújthat szolgáltatásként. Elmondható, hogy nincsen széles körben elfogadott definíció a cloud computing fogalmához, de általánosan elmondható a legtöbb számítási felhőről, hogy a szolgáltatást feliratkozás alapján lehet igénybe venni, úgy, hogy csak a valós erőforrás használatért kell fizetni (*pay-per-use*), amelynek mértéke rugalmasan változtatható, az erőforrások végtelen illúzióját keltve. Az önkiszolgáló interfészeken keresztül virtualizált hardver igényelhető, a valós fizikai konfiguráció a felhasználó elől mindig rejtve marad.

Az IaaS felhő rendszerek egy infrastruktúrát nyújtanak a felhasználó számára, amelyen virtuális gép igényelhető, amelynek különböző paraméterei igény szerint konfigurálhatóak. A főbb értékek:

- *Processzor* (magok száma, sebessége) és *memória* (maximális méret)
- *Lemez*: méret, sebesség, automatikus mentések, további virtuális diszkek.
- *Hálózat*: LAN és WAN beállítások, IP cím DHCP vagy NAT, hozzáférési beállítások virtuális tűzfal segítségével
- *Operációs rendszer*, előre telepített *programok*

Cloudok esetén megkülönböztetünk publikus (*public*), privát (*private*) és hibrid (*hybrid*) megvalósítási modellt. Publikus cloudról beszélünk abban az esetben, ha egy szolgáltatótól vesszük igénybe a virtualizált szolgáltatást interneten keresztül, pay-per-use alapon. Privát cloud esetén az adott cég házon belül valósít meg egy felhő szolgáltatást dolgozói számára, hogy a belső erőforrások kihasználtságát optimalizálja és a kritikus, üzleti adatokat mások elől rejtve tárolja. Olyan cégek esetén, akik privát felhőt használnak, de rendszerüket időnként nagyobb terhelés éri megoldást jelent a hibrid megközelítés használata, amely azt jelenti, hogy nyilvános cloudokból vesznek "kölcsön" kapacitást annak érdekében, hogy a terheltségben jelentkező tüskéket kiegyenlítsék.

Az egyik legnagyobb cloud szolgáltató jelenleg Amazon Web Services, amely számos publikus szolgáltatást nyújt, így többek közt IaaS szolgáltatást (*Amazon Elastic Compute Cloud - EC2*), tárolót (*Amazon Simple Storage Service - S3*) és adatbázis szolgáltatást (*Amazon Relational Database Service - RDS*). Jelen kutatás a privát cloudokra terjed ki, amelyeket egy cég saját szervereire tud telepíteni, hogy így optimalizálja erőforrásainak kihasználtságát. Főleg az open source eszközök kerültek kivizsgálásra, köszönhetően annak, hogy a kevés dokumentáció miatt csak ezek belső architektúrája és működése látható át teljesen, valamint segítségükkel implementálni és tesztelni is lehet az egyes algoritmusokat.

Követelmények

A cloud szolgáltatásokkal szemben támasztott követelmények jelentős részét előre jelzi a cloud computing rendszerek definíciója.

Az egyik legnagyobb előny, hogy a végfelhasználó számára a végtelen erőforrások illúzióját kelti, teljesen **rugalmas**. Az Amazon Web Services esetén ugyanúgy igényelhető egyszerre egy vagy akár több ezer virtuális gép is, amelyeket konfigurálni és kezelni is lehet egységesen. A rugalmas erőforrás használat maga után vonja, hogy a használat alapú számlázáshoz szükséges a használat monitorozása, valamint, hogy az egyes műveletek azonnal végrehajtódjanak.

Az **önkiszolgáló interfész** megléte is kulcsfontosságú elem ehhez, amely jelentős előnye a cloud rendszereknek a hagyományos informatikai infrastruktúrával szemben. A felhasználónak így nincs szüksége külső segítségre, hogy változtathasson az igényelt virtuális hardveres környezeten, vagy igényei szerint konfigurálja azt.

Mivel a felhő felhasználási lehetőségei jelentősen eltérnek egymástól, így elengedhetetlen, hogy az igényelt erőforrások könnyen **testre szabhatóak** legyenek. Ez jelentheti IaaS esetén mind a virtuális gép paramétereit, mind a rá telepített operációs rendszert és annak konfigurációját.

Általános követelménynek tekinthető az **adatbiztonság**, amely magában foglalja az adatok konzisztenciáját és integritását is. Fontos összetevője az adatok biztonságának, hogy csak a megfelelő jogosultsággal rendelkező felhasználó férhessen hozzá az adatokhoz.

Bérelt erőforrások esetén elengedhetetlen a **magas rendelkezésre állás**, amelyet a szolgáltatási szint megállapodás (*Service Level Agreement - SLA*) garantál. Ennek teljesítéséhez elengedhetetlen a rendszer hibatűrése, amely megfelelő tartalék komponensek alkalmazásával és hibák korai detektálásával garantálható. A magas rendelkezésre állás biztosításához feltétel az erőforrások megfelelő allokációja, hogy minél kevesebb szolgáltatási idő essen ki a hardver túlterheltsége vagy a virtuális gépek migrációja miatt. [2]

Jelen kutatás az egyes nyílt forráskódú privát cloud szolgáltatók erőforrásallokációs algoritmusaira tér ki.

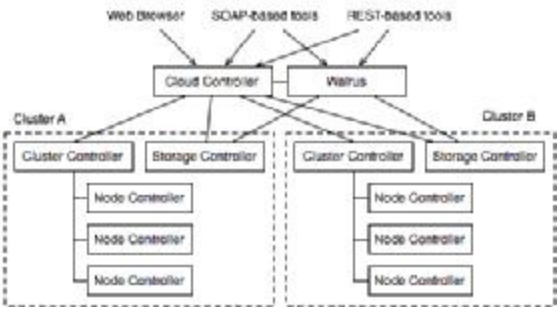
Architektúrák

A vizsgált nyílt forráskódú privát cloudok az Eucalyptus, az OpenStack és a Cloudstack. Ebben a fejezetben ezen cloudok architektúrájának rövid leírása olvasható, amely elengedhetetlen erőforrásallokációs algoritmusaik vizsgálatához.

Eucalyptus

A rendszer magas szintű komponensei különálló web szolgáltatásokként vannak implementálva, amelyek jól definiált interfészeken keresztül érhetőek el. Ezek a következőek:

- Cloud vezérlő (*Cloud Controller*): belépési pontot jelent a felhőbe a rendszer felhasználói és adminisztrátorai számára is. Itt történik meg a felhasználók hitelesítése, valamint a rendszer menedzsment funkcióit is megvalósítja. A cloud egyes csomópontjaitól (*node*) lekérdezett adatok alapján magas-szintű ütemezési döntéseket hoz, amelyeket továbbít az egyes klaszterek felé.
- Felső szintű tároló vezérlő (*Walrus*): Kompatibilis az Amazon S3-mal, így a rendszer képes hibrid felhőként is viselkedni. Kapcsolatban áll a klaszter szintű tároló vezérlőkkel.
- Klaszter vezérlő (*Cluster Controller*): egy privát hálózaton lévő gépek csoportján ellenőrzi a gépeket, ütemezi a virtuális gépek indítását, leállítását, valamint kiválasztja a megfelelő végpontot számukra. Feladata az egyes végpontok adatainak begyűjtése, összegzése, valamint a virtuális gép indítási kérelmek ütemezése az alsóbb komponensek felé.
- Tároló vezérlő (*Storage Controller*): Tároló adatbázis, amelyben virtuális gépek képei valamint felhasználói adatok kerülnek tárolásra.
- Node vezérlő (*Node Controller*): a rendszer minden végpontján futó komponens, amely folyamatosan információt gyűjt a fizikai erőforrásokról, és nyilvántartja az adott hoszt aktuális állapotát. Emellett a node vezérlő hajtja végre a virtuális géppel végzendő konkrét műveleteket is. [3]

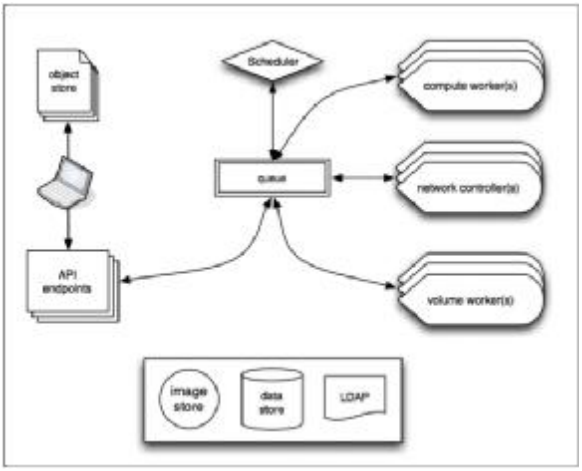


1. ábra: Példa egy lehetséges Eucalyptus telepítés topológiájára

OpenStack

Az OpenStack privát IaaS cloudja három fő komponensből épül fel, az objektumok tárolására szolgáló **Swiftből**, a virtuális gépek tárolását lehetővé tevő **Glance** elnevezésű tárolóból, valamint a számítási erőforrások menedzselését végző **Novából**. A Nova egy menedzsment platform, amelynek feladata az erőforrások, hálózati beállítások és a skálázhatóság menedzselése az OpenStack felhőben. Nem tartalmaz virtualizációs megoldást, viszont drivereket szolgáltat, amelyek segítségével a hoszt operációs rendszeren futó mechanizmusokat kiterjesztik és web API-n keresztül elérhetővé teszik funkcióikat.

A Nova komponensei az *API szerver*, amely interfészt biztosít a külvilág felé, az *üzenetsorok*, amelynek segítségével az egyes belső komponensek közti kommunikáció megtörténik aszinkron módon. Emellett külön egységek végzik el a számítási műveleteket (*Compute worker*), a kötetek kezelését (*Volume worker*), valamint a hálózati konfigurációt (*Network controller*). A rendszer vezérlését az ütemező (*Scheduler*) végzi el. [4]



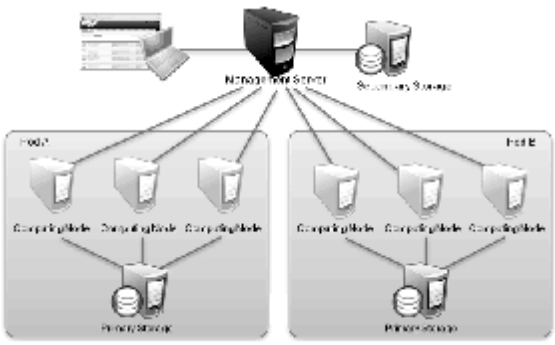
2. ábra: Az OpenStack cloud architektúrája

CloudStack

A CloudStack nyílt forráskódú privát felhő szolgáltatást nyújt. A rendszer egyik fő építőeleme a menedzsment szerver (*Management Server*), amely webes interfészen, illetve egy programozható API-n keresztül nyújt hozzáférést a rendszerhez a felhasználók számára. Ehhez szorosan kapcsolódik egy adatbázis komponens, amely konkrétan egy MySQL adatbázisban tárolja el a rendszer összes konfigurációs beállítását. A rendszer központi részéhez tartozó komponens még a másodlagos tároló (*Secondary Storage*), amelyben a virtuális gépek több felhasználó számára is elérhető telepíthető képei találhatóak.

Logikai csoportosítást jelent a *Pod*, amely a közös virtuális alhálózattal összekötött végpontok gyűjtőhalmaza, amelyen belül további klaszterekbe (*Cluster*) lehet osztani az azonos virtualizációs technológiát használó (azonos *Hypervisor*) fizikai gépeket. Egy pod tartalmaz elsődleges tárolókat (*Primary Storage*), ahol a virtuális gépek lemezei, illetve az azokról készült pillanatképek (*snapshot*) tárolódnak. Az egyes fizikai végpontokon egy ágens fut (*Agent*), amely fogadja a menedzsment szervertől jövő kéréseket, valamint információt szolgáltat a központi gép felé az adott végpont aktuális állapotáról.

A fizikai gépeken futó programok mellett két menedzsment szempontjából elengedhetetlen virtuális gép is fut a rendszerben. Ezek közül az első felelős (*Storage VM*) a rendszer template-ek letöltéséért, feltöltéséért és az ISO képfájlok feltöltéséért a másodlagos tárolóba. A másik rendszer szintű virtuális gép (*Console Proxy VM*) azért felel, hogy konzolos kapcsolatot lehessen létesíteni a menedzsment szerver webes felületére belépve. [5]



3. ábra: CloudStack cloud architektúrája két pod esetén

Algoritmusok

Eucalyptus

Az Eucalyptus cloud forráskódja C-ben íródott és nyílt forráskódjának köszönhetően könnyen értelmezhetőek az erőforrás allokációért felelős algoritmusai.[6] A *Round Robin* algoritmus a klasszikus round robin ütemezésnek felel meg. Sorba halad az egyes erőforrásokon, és mindig a következő olyan gépet választja ki, ahol megfelelő mennyiségű memória, diszk és CPU áll rendelkezésre.

A következő algoritmus a mohó (*Greedy)*, amely a klasszikus első illeszkedés (*First Fit*) algoritmus megfelelője. Az első olyan hosztot választja ki, amin elegendő erőforrás áll rendelkezésre a virtuális gép elindításához.

A mohó algoritmus bővített verziója az energiatakarékos (*Powersave*), amely az első nem alvó hoszton indítja el a virtuális gépet. Ha ilyet nem talál, akkor felébreszt egyet, és azon próbálja meg. Az energiatakarékos üzemmód használata esetén egy bizonyos időküszöbig nem válaszoló gépeket automatikusan elaltatja az klaszter vezérlő komponens energiatakarékossági okokból.

OpenStack

Az OpenStack cloudban különálló komponens felelős az ütemezésért (*nova-scheduler*).[7] Három algoritmus került implementálásra, az egyszerű (*Simple*), esély (*Chance*) és a zóna (*Availability Zones*). Ezek az osztályok Python nyelven készültek, a nyílt forráskódnak köszönhetően megfigyelhető működések. Az OpenStack nem biztosít olyan lehetőséget, hogy a nem terhelt fizikai gépek alvó állapotba kerülhessenek, ahonnan szükség esetén fel lehet őket ébreszteni.

A *Simple* algoritmus a legkevésbé terhelt hosztot (amelyiken a legkevesebb virtuális gép fut) választja ki a lehetséges hosztok listájából. Az algoritmus lefutása kicsit hosszabb időt vesz igénybe, mivel az összes lehetséges hosztot végig kell vizsgálni, viszont előnye, hogy ideális terheléselosztás szempontjából, mivel az összes fizikai gép várhatóan közel hasonlóan lesz terhelve.

A *Chance* algoritmus teljesen véletlenszerűen választ egyet az összes végpont közül. Az algoritmus előnye a sebességében rejlik, valamint abban, hogy megfelelően sok végpont esetén várhatóan jól osztja el a terhelést.

Az *Availability Zones* ugyanúgy működik, mint az esély algoritmus azzal a különbséggel, hogy további paramétere egy zóna, amelyen belül levő hosztok közül kerül véletlen kiválasztásra a lefoglalandó erőforrás, azaz minimálisan befolyásolhatjuk, hogy a virtuális gép hol induljon el.

CloudStack

A CloudStack szolgáltatás Javaban íródott, a megfelelő hoszt allokálásáért felelős osztályok egy specifikus interfész függvényeit implementálják, így könnyen bővíthetőek.[8] A meglevő algoritmusok közül az első az OpenStack *chance* algoritmusához hasonló *RandomAllocator*, amely egy olyan véletlenszerűen választott fizikai gépet választ, amelyen a korábbi foglalások ismeretében található elegendő szabad memória és CPU kapacitás. Ennek egy egyszerűbb változata a *TestingAllocator*, amely le sem ellenőrzi a kiválasztott gép szabad erőforrásait, hanem egyszerűen megpróbálja létrehozni rajta a virtuális gépet.

Az Eucalyptus mohó algoritmusához teljesen hasonló módon működik a *FirstFitAllocator*. Az első olyan hosztot választja ki a virtuális gép számára, amelyen található elegendő kapacitás annak futtatásához.

Kutatási cél

Elmondható, hogy az egyes nyílt forráskódú felhő rendszerekben található erőforrásallokációs algoritmusok igen kezdetlegesek, bővítésre szorulnak, hogy éles környezetben használhatóak legyenek Erre több kutatás is irányult már.[9][10] A megfelelő algoritmus előtt célszerű kiválasztani, az optimalizálandó célt, mivel

ezek egymásnak ellentmondanak, nem létezik olyan megoldás, amely minden paraméter szerint a legjobb lenne.

Amennyiben az a cél, hogy a rendszer **minél előbb kiszolgálja** az erőforrásallokációs kérést, akkor a futtatott algoritmus sebességére kell optimalizálni, úgy, hogy minél kevesebb értéket ellenőrizzen, viszont lehetőleg garantálja, hogy a kiválasztott gépen valóban sikeresen le tudjon futni a kérés.

Amennyiben a rendszerünket érő **terhelést** szeretnénk **kiegyenlíteni**, akkor meg kell tudnunk becsülni a rendszer összes erőforrásának valós terhelését és úgy kell kiválasztanunk az új terhelendő gépet, hogy a becsült új együttes erőforrás foglalás a lehető legkisebb legyen. Ezt elősegíthetjük már futó virtuális gépek migrációjával is, amelynek segítségével jobb eredmény érhető el, de az algoritmus futási ideje és komplexitása megnő.

A harmadik cél a **költségcsökkentés** lehet, amikor a lehető legkevesebb fizikai erőforrás használatára optimalizálunk, hogy a lehető legkevesebb áramot fogyassza a rendszer (a használatlan komponensek alvó állapotban legyenek). Ilyenkor az algoritmus fő célja, hogy a meglevő bekapcsolt szerverek között úgy ossza szét a virtuális gép indítási és lemezfoglalási kéréseket, hogy azok lehetőleg maximálisan ki legyenek használva.

Az egyes algoritmusok teszteléséhez egy CloudStack felhő került konfigurálásra négy fizikai szerverrel. Célunk, hogy erre a rendszerre tudjuk implementálni és tesztelni az egyes célok szerint optimálisnak talált algoritmusokat. Ehhez különböző virtuális gép paraméterekkel és terheltséggel fogunk majd méréseket végezni. Célunk, hogy megfelelő megoldást tudjunk ajánlani minden felhasználási esetre.

Összefoglalás

Jelen kutatás során megvizsgáltuk a jelenleg megtalálható összes megfelelő érettségi állapotban levő nyílt forráskódú IaaS cloud szolgáltatást. Összehasonlítottuk az egyes rendszerek architektúráját, valamint megvizsgáltuk őket erőforrásallokáció szempontjából. Felállítottunk egy CloudStack alapú tesztrendszert, amelyen módosításokat tudunk végezni forráskódjának bővítésével. Célunk saját algoritmusok fejlesztése, implementálása és tesztelése ezen a rendszeren.

Köszönetet mondunk mindazoknak (BME VIK hallgatóinak, egyetemi kutató kollégáknak), akik munkájukkal, tanácsukkal és szakértelmükkel segítették jelen kutatást.

A munka szakmai tartalma kapcsolódik a "Új tehetséggondozó programok és kutatások a Műegyetem tudományos műhelyeiben" című projekt szakmai célkitűzéseinek megvalósításához. A projekt megvalósítását a TÁMOP -4.2.2.B-10/1--2010-0009 program támogatja.

Hivatkozások

[1] J. Staten, S. Yates, F. E. Gillett, W. Saleh: Is Cloud Computing Ready For The Enterprise? (2008, *Forrester Research*)

[2] R. Buyya, J. Broberg, A. Goscinski: Cloud Computing - Principles and Paradigms, (*2011, pp. 3-4 [Buyya et al]*)

[3] Eucalyptus: http://eucalyptus.com/

[4] OpenStack: http://openstack.org/

[5] CloudStack: http://cloudstack.org/

[6] Eucalyptus forráskód: http://open.eucalyptus.com/wiki/eucalyptus-source-code-11

[7] OpenStack forráskód: https://launchpad.net/openstack

[8] CloudStack forráskód: https://github.com/CloudStack/CloudStack

[9] H. Zhong, K. Tao, X. Zhang: An Approach to Optimized Resource Scheduling Algorithm for Open-source Cloud Systems (*ChinaGrid Conference, 2010 Fifth Annual, 16-18 July 2010*)

[10] J. Hu, J. Gu, G. Sun,T. Zhao: A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment (*Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on, 18-20 Dec. 2010*)