Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Intelligens adatfeldolgozás és -elemzés modell alapú támogatása

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

Készítette
Nádudvari György

Konzulens
Gönczy László
tanársegéd

Méréstechnika és Információs Rendszerek Tanszék
Hibatűrő Rendszerek Kutatócsoport

2013. október 25.

Tartalomjegyzék

Kivonat

Informatikai rendszerekből származó adatok elemzésének egyik fontos lépése az ún. feltáró adatelemzés, mely megalapozza a későbbi statisztikai jellegű vizsgálatokat. A feltáró adatelemzés során az adattisztítás és az elemzés eredményességének feltétele annak a szakterületnek az ismerete, amelyből a vizsgált adathalmaz származik, annak érdekében, hogy a megfigyelt jelenségeket helyesen értelmezzük, ill. hatékonyan tudjuk kezelni az adatokat. Ugyanakkor az adatelemző és az adott adatforrás terület szakértője különbözhet. Jogosan merül fel a kérdés, hogy hogyan tudjuk az adatelemző és a szakértő tudását egy helyen, együttesen alkalmazni, elősegítve ezzel az adattisztítás nehézkes műveletének egyszerűsítését, gyorsítását és magának az adatelemzésnek a sikerességét.

Magas szintű szakterületi modellezés egy lehetséges módja ontológiák használata, melyek egyben lehetővé teszik a modellek formális ellenőrzését is. Munkám során arra keresem a választ, hogy egy adott szakterület tudást (pl. metrikák várható korrelációja, topológia kapcsolatok) ontológiában rögzítve és ezt a tudást összekötve a szakterületből származó adathalmazzal, hogyan lehetséges a fentebb említett adattisztítás egyszerűsítése és az adatelemzés során megállapított új tudás felvétele a már létező tudásbázisunkba, hogy ezzel is segítsük a későbbi vagy más által elvégzett elemzéseket az adott területről származó adatokon.

Dolgozatomban virtualizált infrastruktúrák valós mérési adatokon alapuló teljesítményelemzését viszem végig az elkészült eszközzel, amely a tudásbázis tárolására gráf alapú adatbázist, az adatok feldolgozására pedig R statisztikai környezetet használ. Megvizsgálom, hogy a tetszőleges adatforrásból, ill. adatbázis-technológiából származó adatokat hatékonyan tudom-e megtisztítani, előkészíteni az elemzésre, és képes vagyok-e új ismeret feltárására az adatelemzés során.

Mindezek mellett a segédeszköz elkészítése során figyelembe veszem, hogy az tetszőleges más szakterület esetén is alkalmazható legyen, akár más adatelemző eszköz alkalmazásával is. Ezen kívül lehetőséget biztosít az elemzés során vizsgált rendszer (pl. virtualizált infrastruktúra) vagy a mérési módszerek változásának lekövetésére is. Végül bemutatok néhány gyakorlati példát arra, hogyan lehet a módszert alkalmazni más szakterületeken.

Abstract

During the processing and analysis of data coming from information systems exploratory data analysis is an important step establishing further statistic examinations. The knowledge of the domain under analysis is a necessary requirement of the efficient data cleaning and analysis process. However, the data analyst and the domain expert may be different persons.

Therefore the focus of my research is how to utilize the knowledge of the data analyst and the domain expert to the simplicity and speed-up the difficult data cleaning process and to facilitate the data analysis.

High-level domain models can be captured by ontologies which also support the checking of the formal description of these models. During my work I investigate how the knowledge of a given domain (e.g. correlation between metrics, knowledge of topology connections) stored in ontology can support the data process, how further analysis steps can be supported and how the knowledgebase can be extended by information gathered by data analysis.

In my research I evalute the case study of the performance analysis of virtualized infrastructures based on real measurement data. I use graph-based databases to store the knowledge base and the R statistical environment for the data processing. I examine how to clean and prepare data independently from the data source and the database technology to explore new information during the data analysis.

The implemented software is independent from the data representation and can be applied with other analysis tools as well. Moreover it supports the investigation of the effect of changes of the analysed system or the measurement methods by providing traceability mechanisms between system models and measurement data.

1. fejezet

Bevezetés

Az információs társadalmak fejlődésének egyik következménye az előállított adatmennyiség exponenciális növekedése. Egyes források szerint[10] ez a mennyiség naponta 2,5 milliárd gigabyte, amelynek nem csak tárolása, hanem feldolgozása és elemzése is kihívást jelent. Ahhoz, hogy ezeket az adatokat hatékonyan tudjuk elemezni, az elemzéssel új tudáshoz, új információhoz jussunk, új eljárásokat kell keresnünk, amelyek segítik egy részt a nagy adatmennyiség feltáró adatelemzését, másrészt az azt megelőző adattisztítást.

Ahhoz, hogy eredményeket érjünk el az adattisztítás, adatelemzés során, ismernünk kell azt a szakterületet (pl. informatikai infrastruktúra, felhő alapú konfigurációk, termelés vezérlő rendszer), ahonnan a feldolgozandó adatok származnak. Az adatok csak az adott szakterület ismeretével együtt rendelkeznek jelentéssel. Ám az esetek többségében az adatelemző nem az adott szakterület szakértője, így vagy el kell sajátítania a terület ismeretét, ami gyakran nem lehetséges, vagy szakértő segítségét kell igénybe vennie.

1.1. Motiváció

Az adatelemzés fontossága kiemelkedő a döntéstámogatás területén. Az üzleti folyamatokból származó adatok elemzésével optimalizálhatjuk a folyamatot, amely költségcsökkenéshez vezethet. Felhasználói viselkedéssel kapcsolatos adatok elemzése során egyszerűsíthetjük a felhasználók által használt interfészt, ezzel növelve a felhasználók elégedettségét. Historikus teljesítmény adatok vizsgálatával előre jósolhatjuk a rendszerünk kihasználtságát, amely virtualizált környezetben, főleg a cloud computing esetében releváns megtakarítási lehetőségeket nyújthat. A célzott hirdetési rendszerek is egyre pontosabban tudnak a felhasználó érdeklődésének megfelelő ajánlatokat tenni, csupán csak a korábban összegyűjtött adatokból.

Jelenleg is fut olyan kutatási projekt, amely speciális termék optimalizálását célozza meg, az egyes lépések közti összefüggések figyelembevételével. Ebben az esetben az egyes lépések közti sorrend, ill. az elvárt egymásra hatások ismerete fontos lehet a redundáns lépések felismeréséhez.

A weboldalak üzemeltetőinek fontos üzletfejlesztési információ lehet annak nyomon követése, hogy az oldal látogatottsága milyen időszakokban hogyan alakul, vagy hogy milyen az oldal struktúrája, mely aloldalakat látogatják többször. Ezt akár egy tetszőleges weboldal webszerver naplóinak elemzésével megállapíthatjuk.

1.2. Problémafelvetés

A nagy mennyiségű adathalmaz önmagában nem érték, azt valamilyen módszerrel feldolgozni, tisztítani, elemezni kell. Ezen műveletek pontos folyamata függ az adott szakterülettől, hiszen más lehet az adatforrás, más lehet az elemzés célja, más megfigyelést szeretnénk ezáltal igazolni.

Ehhez a szakterületről származó tudást (adatok besorolása, lépések elvárt egymás utánisága, mérési/működési tartományok határai, a rendszerben fellépő hibák elvárt megjelenése az adatok szintjén) célszerű az matematikai elemzés szintje fölötti szakterületi modellben ábrázolni, melynek megfelelő használatával az elemzés paraméterezése támogatható lehet.

Az adatelemzést nagyban segítené, ha rendelkeznénk egy olyan modellel, amely leírja az adott szakterület tudását és ezen modell alapján el tudnánk végezni az akár automatikus, parametrizált adattisztítást és elemzést, majd az új információkat a modellbe visszaírva mi vagy akár más elemzők később újra fel tudnák használni.

Mindemellett egy ilyen modell az adatelemzést végzők közötti kooperációt is segítené, gyorsítaná, hiszen a szakterülettel kapcsolatos, már ismert tudást nem kellene újra és újra felderíteni. Könnyebben tudnánk dimenzió redukciót végezni, és ezzel a feldolgozandó adatmennyiséget csökkenteni, hiszen a modellből kiderülhet, hogy mely adatok között kell lennie összefüggésnek.

1.3. Célkitűzés

Jelen dolgozattal célom annak a vizsgálata, hogy lehetséges-e az adattisztítás, adatelemzés hatékony támogatása olyan adatmodellel, amely az adathalmaz szakterületével kapcsolatos tudásbázist tárolja, azt az adatelemzés folyamatában felhasználja, és bővíti, bizonyos lépések automatikus végrehajtását vezérli.

Egy olyan rendszert készítettem el, amely egy esettanulmányon keresztül mutatja be a módszer működését, és ad-hoc elemzési módszerekkel összehasonlítva bemutatom annak előnyeit.

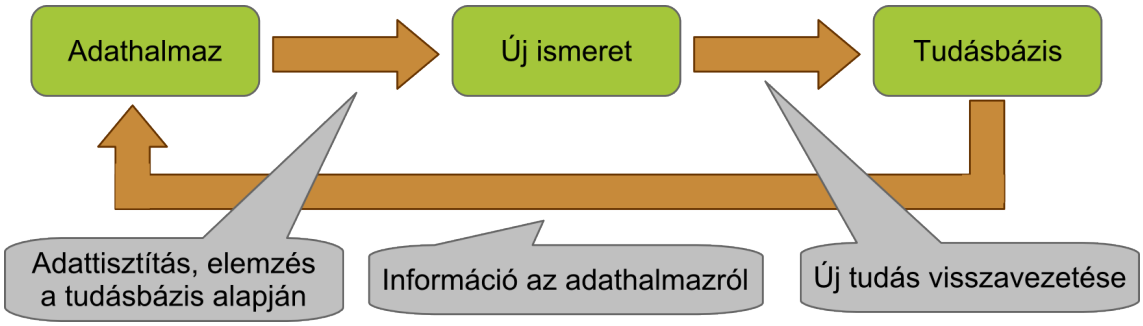
Ehhez elkészítettem egy olyan adatmodellt, amely az adatelemzéshez szükséges információkat tárolja az esettanulmányra, de nem kizárólag arra fókuszálva, kiválasztottam egy már olyan létező technológiát, amely képes annak tárolására, és lehetőséget biztosít annak módosítására is. Az egész rendszer összefogására implementáltam egy vezérlő réteget.

Az esettanulmányhoz kapcsolódó felhasználási esetek közül néhány lényegeset kiválasztottam és azokat az elkészült rendszerrel újra végig vittem, ellenőrizve ezzel annak használhatóságát.

1.4. Megközelítés

A koncepciót mutatja be az 1. ábra. A szakterület specialista tudásbázisban rögzített tudását az adathalmaz tisztítása, elemzése során felhasználjuk, majd az új ismeretet rögzítjük a tudásbázisba, amit később újra felhasználhatunk.

A rendszer használhatóságának demonstrálására kiválasztottam egy esettanulmányt, amely egy létező virtualizált infrastruktúrából származó, valós metrikákat tartalmazó adathalmaz tisztításának és elemzésének lépéseit veszi végig.



1. ábra. A rendszer koncepcionális áttekintése

1.5. A dolgozat felépítése

A bevezető fejezet után egy áttekintés adok azokról a kapcsolódó területekről, amelyeket valamilyen módon érintettem a munkám során. Először az adatelemzésről, annak típusairól és kapcsolódó feladatairól írok. Ezután röviden bemutatom az ontológiát és az ahhoz kapcsolódó technológiákat, olyan megoldásokat keresek, amelyek támogatják annak hatékony tárolását.

A következő fejezetben egy esettanulmányon keresztül mutatom be az elkészült rendszert. Ismertetem a annak architektúráját, az egyes rétegeit. Kitérek az esettanulmány szakterületének, és az ahhoz kapcsolódó adatelemzéssel kapcsolatos használati esetek bemutatására.

Az ez utáni fejezetben vázolom az általam felépített ontológiák szerkezetét és a köztük lévő kapcsolatot. Majd ezután bemutatom a használati esetek végrehajtását és eredményét.

Az utolsó előtti fejezetben kiértékelem a rendszert működését, használhatóságát, skálázhatóságát, és kitérek a továbbfejlesztési lehetőségekre is.

Az utolsó fejezetben összefoglalom az elvégzett munkámat.

2. fejezet

Kapcsolódó szakirodalom

Ebben a fejezetben néhány olyan kutatási témát, alkalmazott technológiát gyűjtöttem össze, amelyek segítenek megérteni az elkészült rendszer koncepcióját, vagy valamilyen más módon (megvalósítás, elképzelés segítése) kapcsolódnak hozzá.

2.1. A feltáró adatelemzés

Az adatelemzés során nagy mennyiségű adatból hasznos és új információt nyerünk ki, majd ezekkel az információkkal következtetéseket tudunk levonni vagy már meglévőket alátámasztani. Napjainkban az adatelemzés fő célja a döntéstámogatás azokban a rendszerekben, folyamatokban, amelyekből a feldolgozott adat származik, vagy amelyekhez valamilyen módon kapcsolódik.

Az adatelemzés egyik módszere az *adatbányászat* (data mining), amely a modellezésre és az ismeretek megszerzésére fókuszál inkább jósló, mint leíró módon. A *big data*[3] szintén az adatelemzéshez kapcsolódik, amely inkább az adatok feldolgozásához szükséges technológiákat jelenti, mint szigorúan az adatok elemzésének módszereit. Az exponenciálisan növekvő adatmennyiségek miatt egyre fontosabb témakör szerepét tölti be.

2.1.1. Az adatelemzés fontosabb lépései

Az adatelemzés fontosabb lépései sorrendben a következők:

1. adatok gyűjtése
2. adatok tisztítása, transzformálása
3. adatok feldolgozása, információk kinyerése

A fenti lépések közül az egyik legnehezebb feladat az adatok megtisztítása. Erre kisebb adatmennyiség esetében manuális módszereket használnak, de nagyobb mennyiségek esetén csak automatikus módszerek alkalmazhatóak.

2.1.2. Az adatelemzés típusai

John Wilder Tukey (1915–2000)[1] amerikai matematikus[2] az adatelemzési módszerek két fő típusát különböztette meg, amelyek a következők:

- megerősítő adatelemzést (CDA - Confirmatory Data Analysis),
- feltáró adatelemzés (EDA - Exploratory Data Analysis).

A megerősítő adatelemzés

Az adathalmazból megállapított új információ általában egy új minta. A megerősítő adatelemzés során két módszert alkalmaznak. Az egyik egy korábbi modell ellenőrzése a talált mintán, amelyet nem használtunk a minta megtalálására. A másik lehetőség a valószínűségszámítás alkalmazása a minta bekövetkezési esélyeinek tesztelésére. Az adatelemzés ezen típusa stabil matematikai és statisztikai tudást igényel[6].

2. ábra. John Tukey (1915–2000)

A feltáró adatelemzés

Feltáró adatelemzésnek nevezzük azt a folyamatot, amikor egy nagy méretű adathalmazból új információkat, következtetéseket próbálunk meg levonni. Ezeket a következtetéseket segíti az adott területtel kapcsolatos háttértudás.

A feltáró adatelemzés egy alesete az interaktív, vizuális adatelemzés. Tukey szerint az adatfeltárás összegző táblázatokat, és alap vizualizációkat használ az adatokban található rejtett minták megtalálására.

Az EDA-nak a CDA-hoz képest más a szerepe, kevesebb matematikai eszközt és tudást igényel. Sok esetben érdemes az EDA során megtalált új ismereteket a CDA-ban használt alaposabb matematikai ellenőrzések alá venni.

2.1.3. Az adattisztítás és elemzés kihívásai

Az adattisztítás és elemzés legnagyobb nehézsége, hogy az adatok önmagunkban nem tartalmaznak információt vagyis egy adathalmazról nem tudjuk megmondani, hogy mit jelent. Ebből következik, hogy egy adott adat értékről önmagában nem feltétlenül tudjuk azt sem eldönteni, hogy az érvényes, értelmezhető, vagy helyes-e.

Ahhoz, hogy egy adathalmazt meg tudjunk tisztítani, vagyis:

- a hibás értékeket
  - eltávolítsuk
  - meg próbáljuk kijavítani
- az egyes értékeket feldolgozható formátumra alakítsuk
- az irreleváns értékeket eltávolítsuk

[1] A róla készült kép forrása: http://en.wikipedia.org/wiki/File:John_Tukey.jpg

[2] Nevéhez fűződik még az FFT algoritmus megalkotásában való részvétel, a box plot és a „bit” fogalmának bevezetése. [27]

ismernünk kell azt a területet, ahonnan az adatok származnak. Ha rendelkezünk ezzel az információval, akkor sikeresen végezhetjük el a tisztítás műveletét.

Az elemzés során szintén azt a tudást használjuk fel, amelyet már a tisztítás során alkalmaztunk. De itt nem elég az adatok jelentésének ismerete, látnunk kell az adatok közötti vélt vagy elvárt összefüggéseket (például topológiai vagy egyéb oksági), és ezen összefüggések alapján vagyunk képesek újabb összefüggések, ismeretek megtalálására. Ezt általánosságban nem támogatják az eszközök, kivéve néhány specifikus területet (pl. gyógyszerészet).

2.2. Tudásbázis modellezése ontológiával

Tudásbázisok modellezésére számtalan lehetőség van. Ezek közül én most az ontológiát fogom bemutatni.

2.2.1. Mi az ontológia?

Az ontológia a számítástudomány és információ tudomány területén olyan tudás reprezentáció, amely egy szakterületről származó fogalmak halmazát és a köztük lévő kapcsolatokat, tulajdonságokat írja le. Az ontológiákat több tudományterületen is alkalmazzák, mivel az információt olyan strukturáltan tárolja, hogy azt számítógépekkel is feldolgozható. Ilyen tudományterület a rendszermérnökség, szoftverfejlesztés, bioinformatika, gyógyszeripar, stb.

Az ontológia hármasokból (triple-ökből) épül fel, amely hármas az alany, állítmány és tárgy. Az ontológia alapegysége az egyed (individual), amely rendelkezhet tulajdonságokkal. Egy egyedet a tulajdonságai alapján halmazokba sorolhatjuk. A tulajdonságok típusaik szerint különbözőek lehetnek (pl. szimmetrikus, tranzitív, funkcionális, stb. tulajdonságok).

Léteznek olyan ún. következtető (reasoner) alkalmazások, amelyek képesek arra, hogy egy létező ontológián következtetéseket tudjanak végrehajtani. Ilyen következtetés lehet például az egyedek halmazokba sorolása tulajdonságaik alapján (ha egy számítógép processzorát rossznak jelölünk valamilyen módon, akkor a számítógépet is automatikusan rossznak jelöli a következtető).

Az ontológiáknak több szabványos leíró nyelve is van, ilyen például az RDF (Resource Description Framework), vagy az arra épülő, kiegészítéseket tartalmazó OWL (Web Ontology Language).

2.2.2. Ontológiák összekapcsolás

Az ontológiák az eltérő szakterületeket névterekkel különböztetik meg. Lehetőség van arra, hogy az ontológiákat összekapcsoljuk, és egy nagyobb ontológiaként tekintsünk rá. Vagyis ha készítünk egy borokat leíró ontológiát, majd pedig egy ételeket tartalmazót, akkor a két ontológiát összekapcsolhatjuk, és ezáltal egy bővebb tudásbázishoz jutunk. Az ilyen összekapcsolások lehetőséget biztosítanak arra, hogy a két ontológiát egyként kezeljük. Például az itt említett borok és ételek ontológiából kis kiegészítéssel előállíthatunk egy olyat, amely azt (is) tartalmazza, hogy mely ételekhez mely borokat ajánlják.

2.2.3. Ontológiák lekérdezése és módosítása

Az ontológiák önmagukban használhatatlanok lennének, ha nem lehetne belőlük a számunka szükséges információt kinyerni. Ehhez készítettek egy szabványos lekérdező és módosító nyelvet, a SPARQL-t.

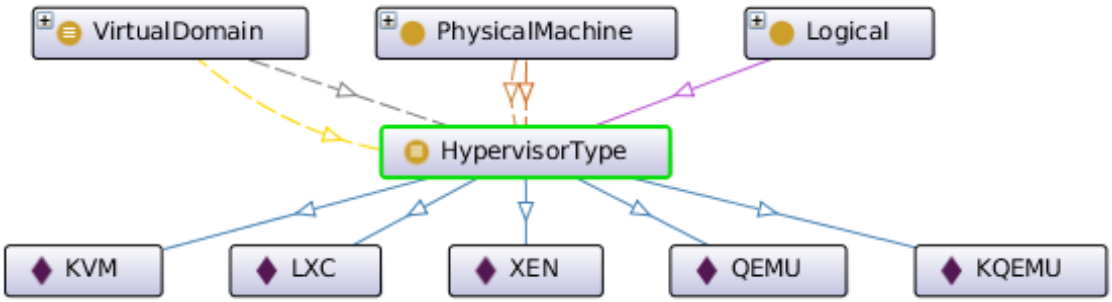
SPARQL

A SPARQL (SPARQL Protocol and RDF Query Language) egy W3C szabványos RDF lekérdező nyelv. Jelenleg az 1.1-es verziónál tart. Szintaktikáját tekintve nagyban hasonlít az SQL nyelvhez.

2.3. Virtualizáció modellezése ontológiával

A virtualizáció és felhő alapú infrastruktúrák megjelenésével megjelent az igény az ezzel kapcsolatos fogalmak rögzítésére és egyértelműsítésére. Mivel az egyes virtualizációs technológiákkal és/vagy valamilyen felhő szolgáltatással foglalkozó cégek eltérően hivatkoznak ugyanazokra a fogalmakra ezért szükséges ezek ontológiába való rögzítése. Persze az eddig létrehozott ontológiák is eltérőek, különböző módon fókuszálnak a téma területeire. Ilyen tipikus fókusz az erőforrások és szolgáltatások leírása, vagy a biztonság[1].

Egy viszonylag részletes ontológiát állítottak elő a torinói egyetem munkatársai[16], amelynek részletét mutatja 3. ábra. Látható, hogy ebben az ontológiában az egyes hypervisor technológiákat is külön kezelik.



3. ábra. A torinói egyetem munkatársai által összeállított ontológia részlete

Érdemes lehet ezt és/vagy az ehhez hasonló ontológiákat felhasználni virtualizált adatelemzést támogató tudásbázis/adatmodell építéséhez.

2.4. Lehetőségek ontológia alapú adatmodell tárolására

A rendszer megvalósításának központi része az adatmodellünk tárolása oly módon, hogy a tároló technológia támogassa több adatmodell összekapcsolását és az adatmodell dinamikus változását. Mivel az adatmodellünk ontológia alapú, amely könnyen ábrázolható egy gráfként és emellett egy három oszlopos táblázatként is (hármasok, vagy triple-ök halmazaként), ezért a specializált RDF adatbázisok mellett a gráf alapú és relációs adatbázisok is szóba jöhetnek.

2.4.1. Relációs adatbázisok

Az ontológia elemei (alany - állítmány - tárgy hármasok) egyszerűen tárolhatók hagyományos relációs adatbázisokban is, de ezzel elveszítjük az ontológia előnyeit, mint a következtetés lehetőségét, vagy a SPARQL lekérdezéseket. Léteznek olyan adatbáziskezelő rendszerek, amelyek próbálnak közelíteni ezekhez az igényekhez, és pl. biztosítanak az SQL lekérdezőnyelv mellé SPARQL lekérdezőnyelvet is, ami a háttérben valójában SQL lekérdezéssé transzformálódik át, tehát nincs natív támogatásuk. Ilyen rendszer az Oracle DB Enterprise Edition[14] és az IBM DB2[11] korábbi verziói.

2.4.2. Gráfalapú adatbázisok

A RDF formátumban tárolt hármasokra úgy is tekinthetünk mint két csomópontra (alany és tárgy) és köztük egy élre (állítmány). Ez a struktúra adja azt a lehetőséget, hogy az ontológiánkat gráfként, gráfok összességeként is tárolhatnánk. Erre adnak lehetőséget a gráf alapú adatbázisok, mint amilyen a a Neo4j is.

Neo4j

A Neo4j egy Java nyelven implementált, kettős licencelésű gráf alapú adatbázis. Előnye, hogy gráf algoritmusokkal dolgozik, így a saját lekérdező nyelve hatékonyabb bizonyos SPARQL lekérdezésekkel szemben (pl. legrövidebb út megtalálása), ugyanakkor a beimportált RDF adatbázison továbbra is lehetőséget biztosít a SPARQL lekérdezések használatára[18]. A Neo4j egy REST alapú webes interfészt biztosít az adatbázis eléréséhez, így minden olyan programozási nyelvvel elérhető, amely képes a HTTP protokollt kezelni.

2.4.3. Triplestore-ok

A triplestore-ok speciális adatbázisok, amelyeket hármasok tárolására és lekérdezésére optimalizáltak. Kezdetekben az ilyen megvalósítások a hagyományos relációs adatbázisokra épültek, ám a ma elérhető, funkciógazdag és hatékony megvalósításokat teljesen az alapoktól írták meg. A rendszer elkészítése során én két implementációt vizsgáltam meg, az OpenLink Software Virtuoso nevű és a Clark & Parsia Stardog nevű szerverét.

Virtuoso

Az OpenLink Virtuoso[17] egy hibrid adatbázisszerver. Relációs, RDF alapú és XML alapú adatmenedzsmentet is támogat. Előnye, hogy transzparensen kezeli az SQL és SPARQL lekérdezéseket, vagyis egy lekérdezéssel férünk hozzá az adatmodellünkhöz és a modellezett adathalmazunkhoz is.

Stardog

A Stardogot[5] a Pellet OWL következető rendszert[4] is fejlesztő Clark & Parsia cég fejleszti, ennek következtében nem véletlen, hogy az eddig felsorolt technológiákkal szemben az egyetlen

olyan implementáció, ami támogatja az ontológia alapú következtetést. Java alapú, egyszerűen beüzemelhető megvalósítás, amely biztosít egy viszonylag jól használható REST-es interfészt is az adatbázisok kezeléséhez és támogatja a SPARQL 1.1-es változatát is. Rendszerem implementálása során végül ezt a triplestore-t használtam.

2.4.4. További lehetőségek

Természetesen a fentebb felsorolt eszközökön kívül több más eszköz is felhasználható a modellünk tárolására. Csak említésképpen a gráf alapú és triplestore adatbázisok közül érdemes lehet kipróbálni az AllegroGraph-ot[8], vagy a nyílt forrású ArangoDB-t[20], Titant[2].

2.5. Virtualizált infrastruktúrák topológiai változásának követése és kezelése

Virtualizált infrastruktúrák esetén gyakori esemény, hogy megváltozik az infrastruktúra topológiája. Ha például egy VMware fürt monitorozás során észreveszi, hogy valamelyik gazdagépének teljesítménye csökken a magas kihasználtság miatt, akkor automatikusan átmozgatja a rajta lévő virtuális gépek egy részét olyan gazdagépekre, amelyek kihasználtsága alacsony. Ezt az eseményt a hagyományos adatgyűjtési technikák esetén nem tudjuk lekezelni, amiből hibás következtetésekre juthatunk az érintett számítógépek elemzése során.

Például ha egy nap a VM1 virtuális gép átkerült a H1 gazdagépre, majd néhány két nap múlva vissza a H2-re, akkor a heti időablakban esetlegesen csak azt látjuk, hogy a H1 kihasználtsága megnőtt, majd lecsökkent, de az okot nem tudjuk beazonosítani. Ezen okokból kifolyólag érdemes lehet az infrastruktúra változását is tárolni az időfüggvényében, és ezt figyelembe venni az adatok elemzésénél.

Ilyen irányban már történtek kutatások a VMware és CMU egyetem részvételével, amelyről az érdeklődő a hivatkozások között található információt[21].

3. fejezet

Az elkészült rendszer bemutatása egy esettanulmányon keresztül

A következőekben az elkészült rendszer felépítését és használatát egy esettanulmányon végig vezetve mutatom be. A választott esettanulmány megoldandó problémái tipikusnak tekinthetők (pl. adatok érvényességének ellenőrzése, adatok közötti összefüggések keresése és ellenőrzése).

3.1. Az esettanulmány bemutatása

3.1.1. Háttér

Az esettanulmány során egy virtualizált infrastruktúrából származó adathalmazon végzünk adattisztítást és elemzést. Az adathalmaz gazda és vendég gépekkel kapcsolatos metrikákat tartalmaz.

A hardveres erőforrások költségeinek csökkenésével megjelent a virtualizáció lehetősége. A virtualizáció nem más, mint erőforrások elosztása különböző virtualizált elemek között, vagyis fizikai elemektől logikai elemek független kezelése. A virtualizáció egyik fajtája a platform virtualizáció, amely esetében a hardverre egy speciális operációs rendszert telepítenek, amely lehetőséget biztosít arra, hogy az általa szolgáltatott virtuális erőforrásokra további operációs rendszereket telepítsünk.

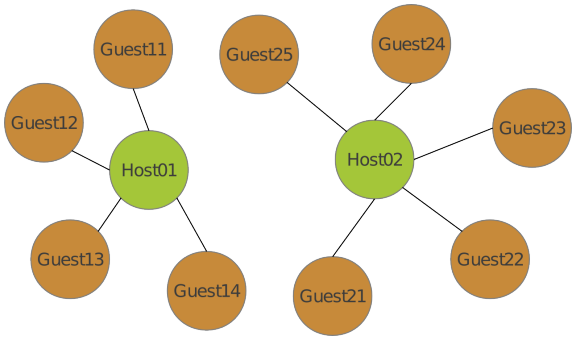
A virtualizáció előnyei például a rugalmas (megfelelő szoftverek esetén akár automatikus) skálázódás vagy a költséghatékony üzemeltetés (nincs, vagy minimálisra csökkenthető a kihasználatlan erőforrások száma). A virtualizáció nélkül nem léteznének a felhőalapú infrastruktúrák, hiszen ezek nagy része virtualizáció használatával kerülnek megvalósításra.

A felhőalapú infrastruktúrák elterjedésével párhuzamosan növekedett a virtualizáció fontossága és egyre fontosabbá vált a felhő szolgáltatók számára ezen infrastruktúrák költséghatékony kihasználása. A kihasználtságot befolyásoló tényezők vizsgálata, az azok közötti összefüggések keresése és az infrastruktúra kihasználtságának, teljesítménybeli problémáinak előrejelzése az adatelemzés izgalmas területe.

3.1.2. A mért infrastruktúra ismertetése

A vizsgált egyetemi infrastruktúra egy 2 szerveres VMware fürt. A szerverek mindegyike 4 magos 2 GHz-es Xeon E5504 CPU-val, 18 GB memóriával rendelkezik. A közös tárhely iSCSI hálózati csatolón keresztül érhető el. Minden szerverben 2 db 1 Gb/s-os hálózati csatoló található, privát belső és nyílt hálózathoz csatlakoztatva.

Az egyik hoston 5, a másikon 4 virtuális gép üzemelt a mérés ideje alatt (lásd 4. ábra). A virtuális gépeken vegyesen Windows és Linux vendég operációs rendszerek futottak. A metrikákat egy PowerShell szkript exportálta ki CSV fájlokba hostonként/VM-enként, napi bontásban. A metrikák mérési intervalluma 20 másodperc, közöttük megtalálhatók processzor, memória, lemez, hálózati és egyéb metrikák is.



4. ábra. A virtualizált infrastruktúra topologikus ábrája

3.1.3. A VMware által használt metrikák áttekintése

Az esettanulmányban megvizsgált VMware-es metrikákat itt nem kerülnek bemutatásra. Az érdeklődő olvasó a hivatkozott források között található referenciákat böngészheti[24], [25], [26], [23], [22]. Az esettanulmány megértéséhez érdemes tudni, hogy a metrikáknak több típusa létezik, adatforrás (csak gazdagépre, csak virtuális gépre, mindkét gép típusra jellemző metrikák) és dimenzió (százalékos, aggregált, stb.) szerinti felosztásban is.

3.1.4. Az esettanulmány során előkerülő használati esetek

A rendszer bemutatása olyan használati esetek felhasználásával történik, amelyek az esettanulmányban fontos lépéseket érintenek valamilyen formában. Ezek általában olyan általános esetek, amelyek több szakterülettel kapcsolatos elemzés során is előkerülhetnek, a gyakorlatból származnak és az adott esettanulmányon kívül számos projekt során összegyűlt problémákra reagálnak:

1. adattisztítással kapcsolatos esetek:
   (a) tudásbázisban megjelölt adatforrásokból származó adatok meglétének ellenőrzése az adathalmazban,
   (b) metrika értéktartományok alapján adatértékek ellenőrzése,
   (c) tudásbázisban rögzített metrika értékkényszerek ellenőrzése és az ellenőrzés eredményének visszaírása a tudásbázisba,
   (d) érvénytelen adatok eltávolítása az adathalmazból,

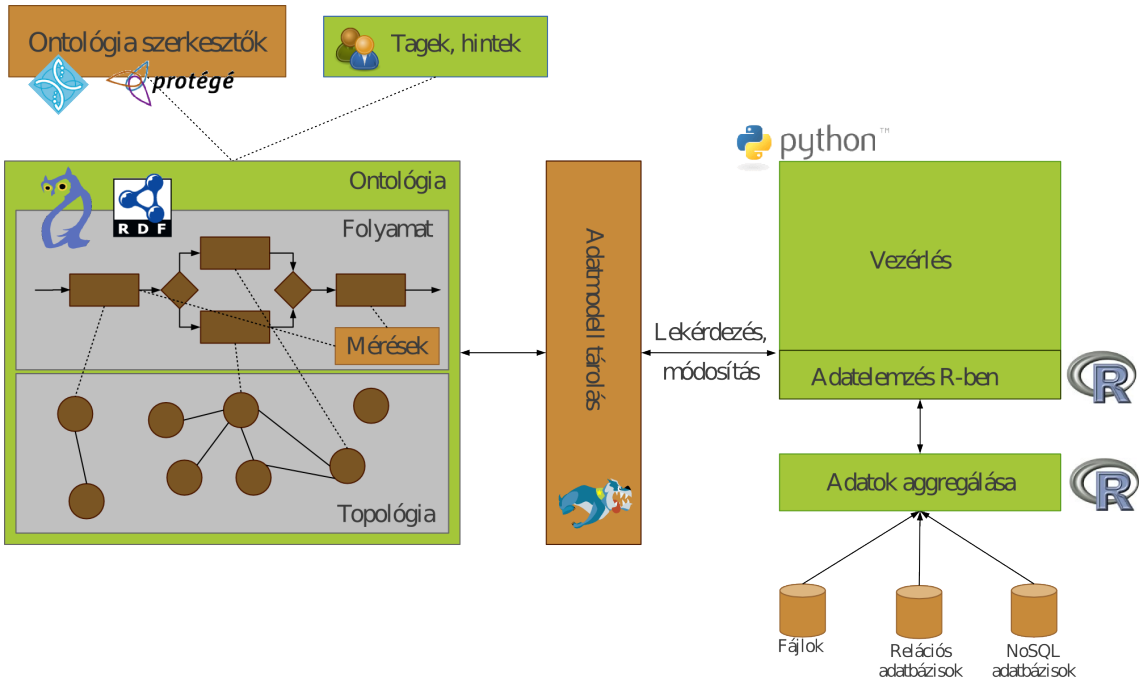
2. adatelemzéssel kapcsolatos esetek:
   (a) tudásbázisban rögzített tartományok alapján az adatforrások kategorizálása,
   (b) tudásbázisban rögzített összefüggések ellenőrzése az egyes metrikák között

Amelyek az esettanulmányra fókuszálva:

1. adattisztítással kapcsolatos esetek:
   (a) minden gazdagépről és virtuális gépről rendelkezünk metrikákkal?,
   (b) a cpu.ready.summation metrika 0 és 20000 közötti, vagyis a mérési intervallumon vett-e fel értéket minden mérési pontban?
   (c) minden olyan metrikánk, amelyet előzőleg „nem lehet ismeretlen” címkével láttunk el ténylegesen teljesíti-e ezt a követelményt?,
   (d) távolítsuk el azokat az értékeket az adathalmazból, amelyek nem meghatározottak/érvényes érték intervallumon kívüliek
2. adatelemzéssel kapcsolatos esetek:
   (a) ha egy adott virtuális gép cpu.ready.summation értékei nagyobbak, mint a mérési intervallum 10%-a, akkor jelöljük meg kritikus teljesítményűnek,
   (b) a cpu.usage.average és cpu.usagemhz.average metrikák között valóban áll-e fenn korreláció?

3.2. Az elkészült rendszer architektúrája

A rendszer kialakításánál elsődleges szempont volt, hogy az egyes komponensek tetszőleges technológiára épülhessenek, így az eredmény egy az 5. ábrán látható réteges architektúra lett, amelyen zöld színel jelöltem az általam elkészített részeket.



5. ábra. Az implementált rendszer architektúrája

A vezérlést Python nyelven implementáltam. Ennek oka, hogy egy egyszerű nyelv, és szá-

mos külső könyvtárral bővíthető. A vezérlőt természetesen tetszőleges olyan programozási nyelven meg lehet valósítani, ami képes az R nyelv vagy más, az adatelemzés rétegében alkalmazott technológia hívására. Az R nyelven implementált kódrészeket, függvényeket Pythonból az rpy2 könyvtár[12] segítségével hívom meg.

Mivel a rendszer független az adatok tárolását végző adatbázis megoldásoktól, ezért a vezérlés és az adatréteg közé került egy köztes, adataggregáló réteg. Ez az implementáció során úgy jelenik meg, hogy az adatbázisból egy RData állományt állítunk elő, amely az R nyelv saját adatstruktúrája, és ezen végezzük el az adatok lekérdezését és módosítását. Természetesen az esetleges változásokat vissza kell írni az adatbázisba. Ez a kialakítás azt is lehetővé teszi, hogy egyáltalán ne használjunk adatbázist, hanem az adatokat fájlokból importáljuk be az RData adatszerkezetbe és később ezt az adatszerkezetet mentsük vagy exportáljuk ki tetszőleges adatformátumba, esetleg adatbázisba.

A vezérlő az adatmodellt egy tároló rétegből éri el. Ez az elérési oda-vissza irányú, tehát az adatmodellt lekérdezhetjük és módosíthatjuk is. Ezt a réteget tetszőleges technológiával megvalósíthatjuk, ami képes SPARQL végpont szolgáltatására. A jelenlegi rendszerben ezt a feladatot a Stardog triplestore szerver látja el.

Az ontológia valójában nem valódi rétege a rendszernek, hiszen az egy tetszőleges ontológia szerkesztővel előállított és az adatmodell tárolóba importált állomány.

Az ábrán nem szerepel a felhasználói interfészt megvalósító réteg. Ennek egyrészt kapcsolódia kell a vezérlő réteghez, másrészt helyettesíthető, pontosabban bővíthető az ontológia réteggel.

4. fejezet

A rendszer működése

4.1. Tudásbázis építése

A tudásbázist a TopQuadrant cég TopBraid Composer Free Edition[19] szoftverével készítettem el. A modell kialakításánál inkább annak egyszerűsége volt a cél, mint részletessége, hogy a koncepció működését minél előbb lássuk. Emellett így az adatelemzéssel párhuzamos tudásbázis fejlesztést is alkalmazhattuk. Sajnos a TopBraid ezen verziójából hiányzik az a funkció, amellyel az ontológiát gráfként tudjuk megtekinteni. Ezt a Stanford Egyetem Protégé[7] nevű alkalmazásával hidaltam át, amelynek az OWLGrpah nevű kiegészítője által előállított gráfokat jelen dokumentációban is használtam.

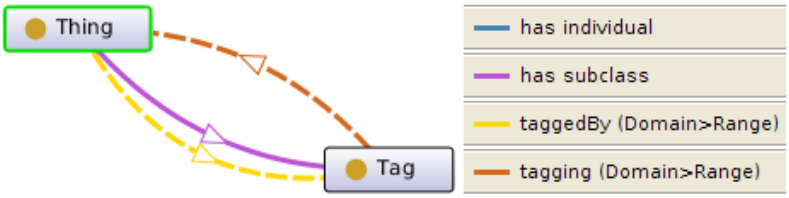
4.2. Az adatmodellünk felépítése

Az ontológiában tárolt adatmodellünk több részre osztható. Egyik központi része a mérés (Measurement) ontológia, amely egy mérés alapvető fogalmait és az azok közötti kapcsolatokat gyűjti össze. Ezt és a későbbi adatelemzést segíti a címke (Tags) ontológia, amely az egyes ontológiák elemeinek felcímkézését, megjelölését segíti. Ezeken az ontológiákon kívül tetszőleges szakterületről származó ontológia lehet a teljes modell része (az esettanulmánynak megfelelően itt egy virtualizációval kapcsolatos ontológia fog bemutatásra kerülni), és egy pillanatkép ontológia, ami valójában az említett ontológiák és vizsgált adathalmazok példánymodellje.

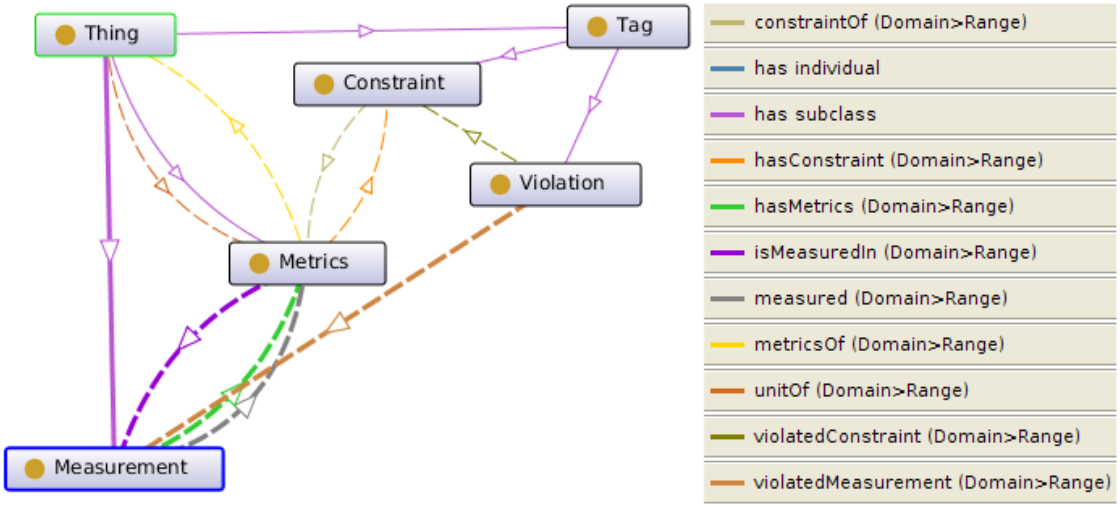
A pillanatkép létjogosultságát az adja, hogy a rendszerben jelenleg még nem megoldott a dinamikusan változó domén kezelése, tehát annak aktuális, statikus képét ez adja, ezenkívül az elemzés során előjöhetnek olyan eredmények, amelyeket nem feltétlenül szeretnénk a szakterület ontológiában rögzíteni üzleti vagy adatkezelési okokból kifolyólag.

4.2.1. A címke ontológia

A címke ontológia (lásd a 6. ábrát) az adatmodellünk legegyszerűbb része. Célja, hogy a későbbi ontológiákban tetszőleges címkéket tudjunk rendelni a fogalmakhoz. Egy címkéhez azt is megadhatjuk, hogy mikor, ki adta hozzá a modellhez.



6. ábra. A címkék ontológia áttekintő ábrája



7. ábra. A mérés ontológia áttekintő ábrája

4.2.2. A mérés ontológia

Az ontológia központi fogalma, ahogy az a 7. ábrán is látható, maga a mérés (Measurement), amelyhez kapcsolódnak a mért metrikák (Metrics). Egy méréshez kapcsolható a mérés ideje és a mért érték, amelyek értéke a metrikához kapcsolódó minimum és maximum értékek közötti tartományból kerülhet ki, és amelyek mértékegységét egy külön tulajdonság adhatja meg (Metrics hasUnit).

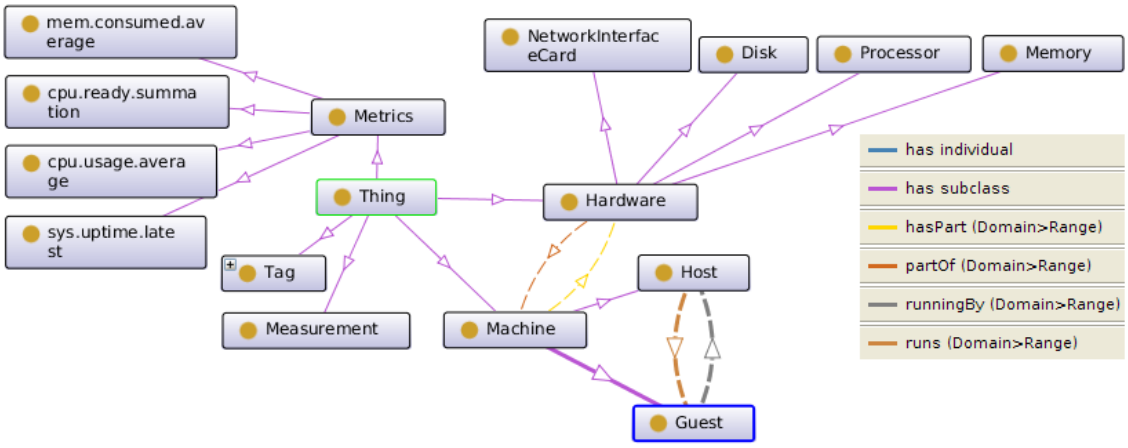
A címkék (Tags) ontológiából származtatva van egy kényszer (Constraint) és a követelmények megsértésére utaló (Violation) címkénk. Minden metrikához tartozhat egy kényszer a minimum és maximum érték korlátok mellett. Ilyen kényszer lehet pl. hogy a metrika mért értéke nem lehet NA (hiányzó, vagy nem értelmezett érték), vagy hogy egy másik értékkel együtt meghatározott érték megfelelő intervallumba esésekor lehet helyes.

Violation címkéket rendelhetünk azokhoz a mérésekhez, amelyek valamely kényszert megsértik. Ezáltal lehetőségünk lehet arra, hogy egy egyszerű lekérdezéssel eldobjuk az adathalmazból azokat a méréseket, amelyek megsértik az érvényességi követelményeket.

4.2.3. A virtualizáció ontológia

A 8. ábrán a virtualizáció ontológia egy kivonata látható. Itt a központi fogalom a gép (Machine), amely attól függően, hogy virtualizált-e, avagy nem (isVirtual) lehet Host vagy Guest. Egy gépnek részei lehetnek hardverek, mint például a pocesszor (Processor), vagy a merevlemez (Disk). A gépnek és a benne található hardverelemeknek lehetnek metrikái, mint pl. egy processzor metrikája

a cpu.ready.summation, vagy a gépé a sys.uptime.latest nevű metrika.



8. ábra. A virtualizáció ontológia áttekintő ábrája

Az általam készített ontológia eltér a 2.3. alfejezetben említett ontológiától. Itt nem térek ki az egyes hypervisorokra, és a virtualizáció is tulajdonságként jelenik meg az egyes fogalmak esetén. Ezenkívül a modellem olyan egyszerű, amennyire csak lehet, hiszen nem egy teljes ontológia összeállítása, hanem a koncepció működésének bizonyítása volt.

4.2.4. Kapcsolat az egyes ontológiák között

Mint ahogy azt már a 2.2.2. alfejezetben is említettem, az ontológiákat össze tudjuk kapcsolni. Az ontológiák ezen tulajdonságára a rendszer erősen épít, hiszen a szakterület tudásbázisát függetlenül akarjuk kezelni a rendszer működéséhez szükséges részektől.

Az esettanulmány során használt virtualizáció ontológia a szakterület ontológiánk, amelyhez kapcsolódik a mérés és címke ontológia. A virtualizált infrastruktúra metrikáit mérjük, és fontossági szinttel vagy a velük kapcsolatos kényszerekkel címkézzük meg.

Az egész tudásbázis központi része a pillanatkép, amelybe valójában rögzítésre kerülnek az adatelemzés során használt vagy megtalált ismeretek, a többi ontológia csak hivatkozásként jelenik meg. Ez alól kivétel lehet az, amikor olyan új ismeretre teszünk szert, amelyet mondjuk a szakterület ontológiájában szeretnénk rögzíteni, mert az ahhoz szorosan kapcsolódik és/vagy másokkal is meg szeretnénk osztani az elkészült ontológiát.

Fontos kiemelni, hogy nincsenek adatpéldányok az ontológiában. Ennek oka, hogy az adatmodellünk mérete jóval kisebb, mint maga a tárolt adatmennyiség. Egy egyszerű adathalmaz esetén is óriásira nőne az ontológiánk. Ha csak egy olyan mérést veszünk, amely 20 másodpercenként 120 metrikáról rögzít adatot, az 1 napi adatmennyiség esetén 518400 elemet jelent az adatmodellünkben, és akkor még nem számoltunk azzal, hogy általában több eszközről rögzítjük a metrikákat. Historikus adatok esetében ez a mennyiség gyorsan milliós nagyságrendűvé válik, és kérdéses, hogy ezek az eszközök milyen sebességgel képesek a lekérdezések eredményét előállítani.

Más lehet a helyzet akkor, ha az adatmodellünkhöz képest elhanyagolható a tárolt adatok mennyisége (pl. hálózatok modellezése), ilyenkor érdemes lehet más megoldási módot keresni.

4.3. Adattisztítási feladatok

Az adat tisztítása során előkerülő feladatokkal már találkozhattunk a 2.1.3. alfejezetben. Itt most ezek közül csak néhányat nézünk meg, amelyek más adathalmaz és szakterület esetén is ismétlődően visszatérhetnek.

4.3.1. Az adatforrások ellenőrzése

Ennél a felhasználási esetnél a a tudásbázisunkban felcímkéztem azokat az elemeket, amelyekből az adatok származnak, vagyis a két gazdagépet és a virtuális gépeket. Ezenkívül egy másik címkével jelöltem azt, hogy az adott adatok hol találhatóak az adathalmazban. A következő SPARQL lekérdezés eredménye az adatforrások halmaza:

```
PREFIX tags: <http://reedcourty.github.com/ontologies/tags#>
SELECT DISTINCT ?source ?type ?class
WHERE {
        ?tag tags:hasName 'Measurement Source'^^xsd:string .
    ?tag tags:tagging ?source .
    ?source rdf:type ?type .
    ?type rdfs:subClassOf ?class .
}
```

Ez a lekérdezés pedig az adathalmaz azon oszlopát adja vissza, amelyben az adatforrásokat találjuk:

```
PREFIX tags: <http://reedcourty.github.com/ontologies/tags#>
PREFIX snapshot: <http://reedcourty.github.com/ontologies/snapshot#>

SELECT DISTINCT ?field
WHERE {
    ?tag tags:hasName 'Measurement Source'^^xsd:string .
    ?tag tags:tagging ?source .
    ?source rdf:type ?type .
    ?type rdfs:subClassOf ?class .
    ?class snapshot:fieldNameInDB ?field
}
```

Ezután a következő R szkriptet felparaméterezve a visszakapott lekérdezési eredményekkel megkapjuk a hiányzó adatforrásokat:

```
get_missing_sources <- function(data, sources_from_model, field_of_source_id) {

    missing_sources <- c()
    data <- data[, c(rep(field_of_source_id))]

    u <- unlist(unique(data))

    print(u)

    for (source in sources_from_model) {
        print(paste(source, source %in% u))
        if (!(source %in% u)) {
            missing_sources <- c(missing_sources, rep(source))
```

```
        }
    }

    return(missing_sources)
}
```

4.3.2. Az egyes kényszerek ellenőrzése

Szinten minden szakterületről származó mérés esetében előfordulhatnak mérési hibák az adathalmazban. Ilyenek lehetnek az érvényes intervallumon kívül eső mért értékek, vagy azok teljes hiánya. Sok esetben ezek elrontják az adatelemzés eredményét, vagy egyes esetekben (pl. korreláció számítás hiányzó értékek esetén) nem is lehetséges.

Az ilyen értékkényszereket egyszerű tárolni a tudásbázisban. A mérés ontológiában minden metrika esetén meg lehet adni egy MaxValue és MinValue tulajdonságot (ahogy azt a 4.2.2. alfejezetben is említésre került), amelyet egy egyszerű SPARQL lekérdezéssel nyertem ki az adott metrikáról, majd az R nyelv segítségével előállítottam egy olyan adathalmazt, amelyből elhagytam azokat a méréseket, amelyek értéke ezen intervallumon kívülre esett.

A hiányzó mérési értékek kezelését ettől eltérő irányból közelítettem meg. Azokat a metrikákat, amelyek értéke nem lehet nem meghatározott (NA), egy IsNotNA címkével látjuk el, majd később lekérdezéssel ezeket a címkéket fogjuk megkeresni.

A címkék hozzáadását a következő SPARQL lekérdezéssel lehet elvégezni:

```
PREFIX tags: <http://reedcourty.github.com/ontologies/tags#>
PREFIX measurement: <http://reedcourty.github.com/ontologies/measurement#>
PREFIX virtualization: <http://reedcourty.github.com/ontologies/virtualization#>
INSERT {
    <http://reedcourty.github.com/ontologies/snapshot#IsNotNA> rdf:type tags:Tag .
    <http://reedcourty.github.com/ontologies/snapshot#IsNotNA> tags:addedAt "2013-10-10
        T09:54:23" .
    <http://reedcourty.github.com/ontologies/snapshot#IsNotNA> tags:addedBy "reedcourty"
         .
    <http://reedcourty.github.com/ontologies/snapshot#IsNotNA> tags:hasComment "This
        metrics values should not NA"^^xsd:string .
    <http://reedcourty.github.com/ontologies/snapshot#IsNotNA> tags:hasName "Is not NA"
        ^^xsd:string .
    <http://reedcourty.github.com/ontologies/snapshot#IsNotNA> tags:tagging ?metrics .
} WHERE {
    ?metrics rdfs:subClassOf measurement:Metrics .
    ?metrics measurement:metricsName "{0}"^^xsd:string .
}
```

A WHERE feltétel utolsó sorában található „{0}” karakterek helyére kerül annak a metrikának a neve, amelyikhez hozzá akarjuk rendelni az adott címkét.

Ezután az ezzel a címkével rendelkező metrikákat a következő lekérdezéssel gyűjtöttem össze:

```
PREFIX tags: <http://reedcourty.github.com/ontologies/tags#>
SELECT ?metrics
WHERE {
    ?tag rdf:type tags:Tag .
    ?tag tags:hasName "Is not NA"^^xsd:string .
```

```
    ?tag tags:tagging ?metrics
}
```

A visszakapott eredmény alapján le kérdezhetem a metrika adathalmazban szereplő azonosítóját, majd az ahhoz tartozó értékekre az R-ben egy egyszerű paranccsal ellenőriztem, hogy teljesül-e a feltétel.

4.4. Adatelemzés

A következő részben a rendszer által támogatott adatelemzéssel kapcsolatos használati eseteket mutatom be.

4.4.1. Az infrastruktúra processzor teljesítményének vizsgálata

Az esettanulmány cpu.ready.summation metrikája jól indikálja egy virtualizációs infrastruktúra kihasználtságát, és ebből következően az esetleges teljesítménybeli problémákat. A metrika értékeinek eloszlása alapján kategóriákba tudjuk sorolni a gépünket a teljesítménye szerint. Ha egy adott virtuális gépről származó cpu.ready.summation értékek nagysága nem haladja meg a mérések között eltelt idő 5%-át, akkor nincs probléma a vendéggép kihasználtságával, a rajta futó virtuális gépek között jól van szétosztva a futási idő. Ellenben ha ez az érték 5-10% közötti, akkor érdemes lehet átrendezni gazdagépek közötti virtuális gép elosztás. 10% feletti értékek esetén a rendszer működése már kritikus kategóriába sorolható.

A rendszerben a cpu.ready.summation metrikát felcímkéztem egy intervallum címkével, amely megjegyzésben tartalmazza a három intervallumot, a hozzájuk tartozó értékhatárokat és egy kategóriát. Ezt az intervallum címkét a következő lekérdezéssel kérdeztem le:

```
<http://reedcourty.github.com/ontologies/measurement#>
PREFIX tags: <http://reedcourty.github.com/ontologies/tags#>
SELECT ?comment
WHERE {
    ?metrics rdfs:subClassOf measurement:Metrics .
    ?metrics measurement:metricsName "cpu.ready.summation"^^xsd:string .
    ?metrics tags:taggedBy ?tag .
    ?tag tags:hasName "Important Metrics"^^xsd:string .
    ?tag tags:hasComment ?comment .
}
```

A visszakapott értéket a vezérlőben feldolgozom és egy R függvénynek adom át, amely egy virtuális gép azonosító - kategória értékpárokkal tér vissza. Ezután ezek alapján „Megfelelő”, „Elfogadható” és „Kritikus” címkékkel látom el az adatmodellben szereplő virtuális gépeket.

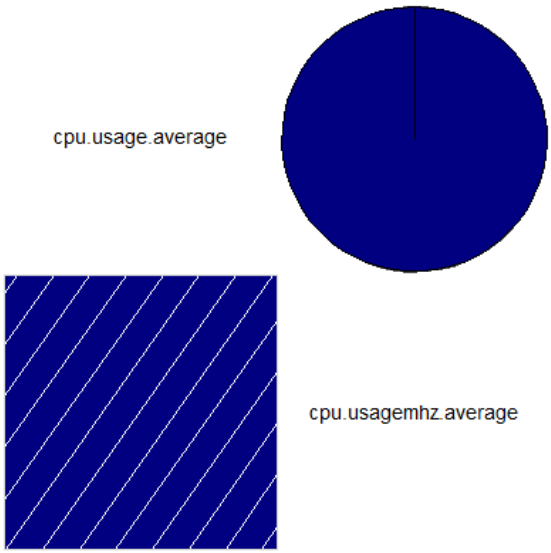
4.4.2. Összefüggések meglétének ellenőrzése az egyes metrikákon

Ha a tudásbázisban rendelkezünk a metrikák közötti összefüggésekre vonatkozó információval, akkor annak meglétét az adathalmazon is ellenőrizhetjük. Példánkban feltételezzük, hogy a cpu.usage.average és cpu.usagemhz.average metrikák között valamilyen korreláció van, így azokat

megjelöltük egy közös DependencyId01 címkével. A következő SPARQL lekérdezéssel megkapjuk eredményül azokat a metrikákat, amelyek között a DependencyId01 összefüggés van:

```
PREFIX tags: <http://reedcourty.github.com/ontologies/tags#>
SELECT ?metrics
WHERE {
    ?metrics rdfs:subClassOf measurement:Metrics .
    ?tag rdf:type tags:Tag .
    ?tag tags:tagging ?metrics .
    ?tag tags:hasName "DependencyId01"^^xsd:string .
}
```

Esetünkben ez a két említett metrika lesz. Ezeket átadjuk egy függvénynek, amely a metrikák által meghatározott értékekből korrelációs diagramot rajzol. Ennek eredménye látható a 9. ábrán, amelyről megállapíthatjuk, hogy a két metrika között tényleg létezik valamilyen (időbeli, számítási, topológiai vagy oksági) összefüggés.



9. ábra. A cpu.usage.average és cpu.usagemhz.average metrikák közötti korreláció

4.5. Új ismeretek, eredmények visszaírása a tudásbázisba

Lehetőségünk van az új ismeretek rögzítésére is az adatmodellbe. Például ha rájövünk, hogy két metrika között korreláció van (pl. minden metrikát páronként megvizsgáltunk és eredményül azt kaptuk, hogy a cpu.usage.average és cpu.usagemhz.average metrikák között valamilyen összefüggés van), akkor azt egy hasonló SPARQL lekérdezéssel felvehetjük a tudásbázisba:

```
PREFIX tags: <http://reedcourty.github.com/ontologies/tags#>
INSERT {
    <http://reedcourty.github.com/ontologies/snapshot#Dependency> rdf:type tags:Tag .
    <http://reedcourty.github.com/ontologies/snapshot#Dependency> tags:addedAt "
        2013-10-25T09:54:23" .
```

```
    <http://reedcourty.github.com/ontologies/snapshot#Dependency> tags:addedBy "
        reedcourty" .
    <http://reedcourty.github.com/ontologies/snapshot#Dependency> tags:hasComment "
        Dependency between {0} and {1}"^^xsd:string .
    <http://reedcourty.github.com/ontologies/snapshot#Dependency> tags:hasName "
        Dependency"^^xsd:string .
    <http://reedcourty.github.com/ontologies/snapshot#Dependency> tags:tagging ?metrics1
         .
    <http://reedcourty.github.com/ontologies/snapshot#Dependency> tags:tagging ?metrics2
         .
} WHERE {
    ?metrics1 rdfs:subClassOf measurement:Metrics .
    ?metrics1 measurement:metricsName "{0}"^^xsd:string .
    ?metrics2 rdfs:subClassOf measurement:Metrics .
    ?metrics2 measurement:metricsName "{1}"^^xsd:string .
}
```

Természetesen a „{0}” és „{1}” paraméterek helyére a két metrika neve kerül.

5. fejezet

Kiértékelés

5.1. A rendszer funkcionalitása

Az elkészült rendszer támogatja azokat a használati eseteket, amelyek megvalósítását célul tűztem ki. A rendszer képes a tudásbázist tároló adatmodellt lekérdezni, és a lekérdezések eredménye alapján el tud végezni bizonyos adattisztító és adatelemző műveleteket, vagy az elemzések eredménye alapján módosítani tudja a tudásbázist.

5.2. A rendszer használhatósága más esettanulmányon

A rendszer koncepciójának és implementációjának kialakításakor törekedtem arra, hogy tetszőleges szakterületről származó adathalmazzal használható legyen. A következőekben összegyűjtöttem néhány másik területet is, ahol eredményesen alkalmazhatjuk a rendszert.

5.2.1. Speciális termékek gyártási és ellenőrzési folyamata

Ahogy azt már az 1.1. alfejezetben érintettem egy ilyen terület esetében az egyik cél az, hogy az időben és helyben eltérő gyártási folyamatlépések közti összefüggések feltárhatóak legyenek.

Az adatok kielemzésének legnagyobb nehézsége, hogy a szakterületet csak az abban dolgozó szakemberek ismerik. Egy adott termék életútját végig kell tudnunk követni a gyártási folyamatból származó adatokon. Ezt egyszerűsítheti, ha a folyamatot rögzítettük, pontosabban az adatelemző szempontjából, rögzítették számunkra. Ehhez az itt bemutatottaktól különböző, a gyártási folyamatok leírását támogató ontológia szükséges.

Egy ilyen folyamat kimenete akkora adatmennyiséget jelenthet, hogy már big data problémákkal szembesülhetünk. Ennek egyik megoldása lehet a dimenzió redukció, amelynek célja olyan adathalmazok felderítés, amelyek más adathalmazokkal úgy függnek össze, hogy elhagyhatóak. Az ilyen összefüggések elemzési előtti ismerete, és azok rögzítése gyorsíthatja az adatelemzés folyamatát. Másrészt a szakértői tudást adatokhoz kapcsolása segíthet ok-okozati kapcsolatokon keresztül csökkenteni az adatfeldolgozási igényt.

5.3. A rendszer bővíthetősége, továbbfejleszthetősége

A rendszer struktúrájának köszönhetően könnyen módosítható, bővíthető, a jelenlegi rétegek könnyen lecserélhetők. A homogenitást lehetne növelni azzal, ha a jelenlegi R nyelven megvalósított részeket Python nyelven, vagy a vezérlőt teljes egészben R nyelven implementálnánk. Néhány továbbfejlesztési lehetőséget sorolok fel a következő alfejezetekben.

5.3.1. Időben változó tudásbázis használat

A bemutatott esettanulmány során előkerülhet az a probléma, amelyet már a 2.5. alfejezetben is említettem, vagyis hogy a jelenlegi rendszerünk nem támogatja az adatmodellünk változását. Ahhoz, hogy ezt kezelni tudjuk, valamilyen módon módosítani, bővíteni kell majd az ontológiát.

5.3.2. A tudásbázist reprezentáló ontológiák fejlesztése

Az elkészült ontológiák közel sem teljesek (valójában a nyílt világ szemantika miatt soha nem is lehetnek). Érdemes lehet új megoldásokat kitalálni arra, hogy hogyan tudjuk általánosítani a mérés és címkézés ontológiát. Egy fontos kérdés, hogy mi lehet az a minimális munka, amelyet el kell végeznünk más szakterület ontológiák saját rendszerünkkel való összekapcsolásakor. Számos kutatás irányul arra, hogy szakterület specifikus modelleket hogyan lehetne automatikusan előállítani, amelyek között az ontológia is felmerül, mint egy lehetséges megoldás[9].

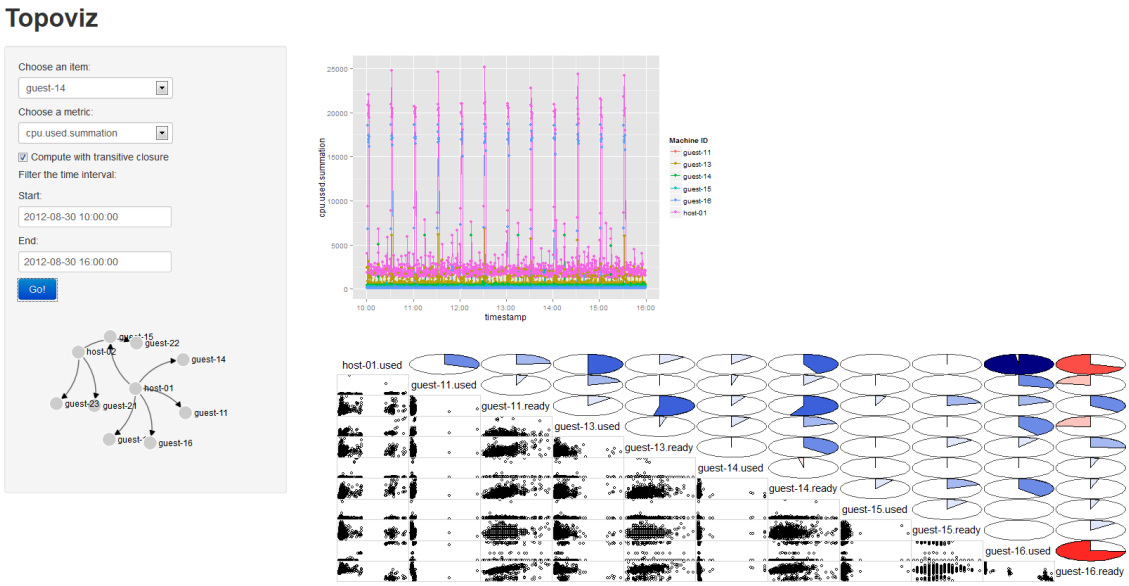
5.3.3. Felhasználói felület fejlesztése

A rendszer jelenlegi állapotában nem végfelhasználóknak szánt termék, csak egy koncepciót bemutató eszköz. Ezért az egyik legnyilvánvalóbb továbbfejlesztési lehetőség a felhasználói felület fejlesztése. Akár egy egyszerű, böngészőből elérhető kezelőfelületet is létre lehet hozni, amely a tudásbázis alapján jelenít meg bizonyos vezérlőeszközöket. Például egy legördülő menü felsorolja a tudásmodellből kiolvasható össze metrikát, majd egy időintervallum megadása után kirajzolja a kiválasztott metrika értékeket a megadott időintervallumban, mint ahogy az a 10. ábrán is látható, amely egy korábbi munkám eredménye.

Az eszköz egy Shiny nevű projektre épül, amelynek célja, hogy egy interaktív, webes felületet adjon a vizuális adatelemzéshez és megjelenítéshez[15]. Ez a későbbi munkám egyik fő iránya lehet.

5.3.4. Ontológia közvetlen szerkesztése

A tudásbázis kezdeti szerkesztése egyelőre nem a rendszer része, hanem azt egy külön eszközzel tudjuk elvégezni (mint ahogy azt a 4.1. alfejezetben is láthattuk). Egyszerűsítené az adatelemző és a szakterület specialista munkáját, ha a rendszerhez közvetlenül kacsolódnak egy ontológia szerkesztő eszköz. Ez által a tudásbázisunkat közvetlenül tudnák szerkeszteni, és nem lenne szükség az ontológiák exportálására/importálására a munka elvégzéséhez. Sajnos egy ilyen szerkesztő elkészítése igen nagy feladat lehet.



10. ábra. Egy lehetséges felhasználói felület

5.3.5. Az adattisztítás, elemzés folyamatának kezelése a rendszerben

Az adattisztítás és elemzés folyamata gyakran tartalmaz ismétlődő lépéseket is az egyediek mellett. Érdemes lehet megvizsgálni, hogy ezt a folyamatot tudjuk-e valamilyen módon ontológiába rögzíteni, majd az ismétlődő folyamatokat automatizáltan végrehajtani, és az ontológiába rögzíteni, hogy azokat már végrehajtottuk. Ezáltal nyomon követhető lehet a folyamat más elemzők által is.

5.4. A rendszer skálázhatósága

A rendszer teljes skálázódásának tesztelésére nem került sor, de a kialakított architektúra lehetővé teszi, hogy az egyes rétegek skálázódásával külön foglalkozzunk. Természetesen kérdéses, hogy egy ilyen rendszer esetében érdemes-e egyáltalán a skálázódással foglalkozni, hiszen várhatóan az adattisztításnak és elemzésnek lesz a legnagyobb teljesítmény igénye, mivel az adatpéldányok nincsenek közvetlenül az ontológiában tárolva.

5.4.1. Skálázhatóság kérdése az adatmodellt tároló réteg esetében

Az adatmodell tároló réteg feladatát elvégző szerverek szolgáltatásként támogatják a megfelelő skálázódást. A már említett Neo4j esetében külön létezik terhelés elosztást, nagy rendelkezésre állást és adat elosztást végző szolgáltatás is[13]. A Virtuoso szerverrel kapcsolatos teljesítmény méréseket a termék honlapján is találhatunk[1]

5.4.2. Az adatelemző réteg skálázódása

A rendszer egészét tekintve ez a réteg igen erőforrás igényes. Alapvető eszközökkel nagy mennyiségű adatok esetében korlátokba ütközhetünk, de szerencsére az esetünkben itt alkalmazott R

[1] http://www.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSScale

nyelv rendelkezik olyan kiegészítő csomagokkal, amelyekkel kezelhetőek az ilyen esetek[2].

[2] Érdemes lehet megtekinteni a http://r-pbd.org/ oldalt

6. fejezet

Összefoglalás

Dolgozatomban arra kerestem a választ, hogy hogyan lehet egy szakterület modellezett tudása alapján elősegíteni a feltáró adatelemzés műveletét. Ezzel olyan problémát kívántam megoldani, melyre a ma elterjedt eszközök nem adnak általános javaslatot. Mindezt rugalmasan bővíthető, potenciálisan jól skálázódó modellel oldottam meg. A rendszer kipróbálása során virtualizált IT infrastruktúrák területéről származó, valós mérési adatokat használtam fel.

Először bemutattam azokat a témákat, amelyek kapcsolódnak az általam vizsgált területhez.

Ezután bemutattam a rendszer felépítését, ismertettem a felhasznált esettanulmányt.

A következő fejezetben részletesen ismertettem néhány fontosabb adattisztítási és adatelemzési használati esetet. Megmutattam az elkészült rendszer és az esettanulmány tudásbázisának felépítését.

Az utolsó fejezetben értékeltem az elkészült rendszer mind használhatóság, mind skálázhatóság alapján, és lehetőségeket mutattam annak továbbfejlesztési irányaira is.

Az általam bemutatott módszer alkalmas lehet arra, hogy nagyméretű, ill. sokféle adatból származó adathalmaz esetén is könnyítse az adatok kinyerésének/elemzésének feladatát azáltal, hogy az adatokra ill. a rendszermodellre vonatkozó szakértői tudást összekapcsolja az adatelemzés lépéseivel.

Ábrák jegyzéke

Táblázatok jegyzéke

Hivatkozások

[1] D. Androcec, N. Vrcek, and J. Seva. Cloud Computing Ontologies: A Systematic Review. In *MOPAS 2012, The Third International Conference on Models and Ontology-based Design of Protocols, Architectures and Services*, 2012.
URL: http://www.thinkmind.org/download.php?articleid=mopas_2012_1_20_50018
(utolsó hozzáférés: 2013.10.24.).

[2] Aurelius. Titan: Distributed Graph Database (weboldal), 2013.
URL: http://thinkaurelius.github.io/titan/
(utolsó hozzáférés: 2013.10.25.).

[3] Bodnár Ádám, HWSW.hu. Értetlenség és érdektelenség övezi a big datát. URL: http://www.hwsw.hu/hirek/50004/oracle-big-data-hadoop-mapreduce.html, 2013. Utolsó hozzáférés: 2013.05.21.

[4] Clark & Parsia, LLC. Pellet: OWL 2 Reasoner for Java (weboldal), 2013.
URL: http://clarkparsia.com/pellet
(utolsó hozzáférés: 2013.10.25.).

[5] Clark & Parsia, LLC. Stardog: The RDF Database. (weboldal), 2013.
URL: http://stardog.com/
(utolsó hozzáférés: 2013.10.25.).

[6] D. Conway and J. White. *Machine Learning for Hackers*. Case studies and algorithms to get you started. O'Reilly Media, Incorporated, 2012.

[7] Stanford Center for Biomedical Informatics Research. The Protégé Ontology Editor and Knowledge Acquisition System (weboldal). URL: http://protege.stanford.edu/, 2013. Utolsó hozzáférés: 2013.10.25.

[8] Franz Inc. AllegroGraph RDFStore Web 3.0's Database (weboldal), 2013.
URL: http://www.franz.com/agraph/allegrograph/
(utolsó hozzáférés: 2013.10.25.).

[9] László Gönczy and István Dávid. Ontology-supported design of domain-specific languages: A complex event processing case study. In *Advances and Applications in Model-Driven Engineering*, pages 106–133. IGI Global, 2013.

[10] IBM Corporation. IBM Big Data - What is Big Data (weboldal), 2013.
URL: http://www.ibm.com/big-data/us/en/
(utolsó hozzáférés: 2013.10.24.).

[11] IBM Corporation. IBM DB2 NoSQL Support (weboldal), 2013.
URL: http://www-01.ibm.com/software/data/db2/linux-unix-windows/nosql-support.html
(utolsó hozzáférés: 2013.10.24.).

[12] Gregory R. Warnes Laurent Gautier, Walter Moreira. RPy - simple and efficient access to R from Python (weboldal), 2013.
URL: http://rpy.sourceforge.net/rpy2.html
(utolsó hozzáférés: 2013.10.25.).

[13] David Montag. Understanding Neo4j Scalability. Technical report, Neo Technology Inc., 2013. január. Utolsó hozzáférés: 2013.10.23.

[14] Oracle Corporation. Oracle Database 12c Enterprise Edition (weboldal), 2013.
URL: http://www.oracle.com/us/products/database/enterprise-edition/overview/index.html
(utolsó hozzáférés: 2013.10.25.).

[15] RStudio, Inc. Rstudio - shiny (weboldal). URL: http://www.rstudio.com/shiny/, 2013. Utolsó hozzáférés: 2013.10.25.

[16] Jacopo Silvestro, Daniele Canavese, Emanuele Cesena, and Paolo Smiraglia. A unified ontology for the virtualization domain. In *On the Move to Meaningful Internet Systems: OTM 2011*, pages 617–624. Springer, 2011.

[17] OpenLink Software. OpenLink Virtuoso Universal Server (weboldal), 2013.
URL: http://virtuoso.openlinksw.com/
(utolsó hozzáférés: 2013.10.25.).

[18] Davy Suvee. Storing and querying RDF data in Neo4j through Sail. URL: http://datablend.be/?p=411, 2011. Utolsó hozzáférés: 2013.10.22.

[19] TopQuadrant. Products - TopBraid Composer (weboldal), 2013.
URL: http://www.topquadrant.com/products/TB_Composer.html
(utolsó hozzáférés: 2013.10.25.).

[20] triAGENS GmbH. ArangoDB - The universal free and open source NoSQL database (weboldal), 2013.
URL: http://www.arangodb.org/
(utolsó hozzáférés: 2013.10.25.).

[21] Ilari Shafer (Carnegie Mellon University), Inc.) Snorri Gylfason (VMware, and Gregory R. Ganger (Carnegie Mellon University). vquery: A platform for connecting configuration and performance. *VMware LABS*, december 2012.
URL: http://labs.vmware.com/academic/publications/vquery-vmtj-winter2012

vagy http://www.pdl.cmu.edu/PDL-FTP/CloudComputing/vQuery.pdf
(utolsó hozzáférés: 2013.05.24.).

[22] VMware Community. Understanding VirtualCenter Performance Statistics, 2011.
URL: http://communities.vmware.com/docs/DOC-5230
(utolsó hozzáférés: 2013.10.25.).

[23] VMware Community. vCenter Performance Counters, 2011.
URL: http://communities.vmware.com/docs/DOC-5600
(utolsó hozzáférés: 2013.10.25.).

[24] VMware, Inc. VMware.com - CPU Counters. URL: http://www.vmware.com/support/developer/vc-sdk/visdk400pubs/ReferenceGuide/cpu_counters.html, 2009. Utolsó hozzáférés: 2013.10.25.

[25] VMware, Inc. VMware.com - Disk I/O Counters. URL: http://www.vmware.com/support/developer/vc-sdk/visdk400pubs/ReferenceGuide/disk_counters.html, 2009. Utolsó hozzáférés: 2013.10.25.

[26] VMware, Inc. VMware.com - Network Counters. URL: http://www.vmware.com/support/developer/vc-sdk/visdk400pubs/ReferenceGuide/network_counters.html, 2009. Utolsó hozzáférés: 2013.10.25.

[27] Wikipedia. John Tukey. URL: http://en.wikipedia.org/wiki/John_Tukey, 2013. Utolsó hozzáférés: 2013.05.21.

7. fejezet

Függelékek

7.1. Függelék - Néhány VMware rendszerben előforduló metrika leírása

7.1. táblázat. Néhány CPU-val kapcsolatos metrika

Metrika	Leírás	Mértékegység
cpu.swapwait.summation	A 20 milliszekundumos ablakból mennyi időt tölt a VM várakozással (munkavégzés nélkül) amiatt, hogy nem tudja memóriába tölteni a működéséhez szükséges adatokat.	milliszekundum
cpu.idle.summation	A 20 milliszekundumos ablakból mennyi időt tölt a VM idle (vagyis futás nélküli) állapotban.	milliszekundum
cpu.ready.summation	Megmutatja, hogy egy virtuális gép az időablak hány százalékában volt olyan állapotban, hogy ugyan futásra készen állt, de nem került futtatási állapotba a fizikai processzoron. Az érték függ a virtuális gépek számától és azok CPU terheltségétől (loadjától).	milliszekundum
cpu.wait.summation	Értéke megadja, hogy a teljes CPU időhöz viszonyítva mennyi időt töltött a virtuális gép várakozó állapotban.	milliszekundum
cpu.run.summation	Megadja, hogy a VM az idő hány százalékát töltötte futási állapotban. (Származtatott érték: 100% = run + ready + wait (+ costop)	milliszekundum
cpu.usage.average	Az aktívan használt virtuális CPU értéke a teljes elérhető CPU-ra nézve. Ez az átlagos kihasználtsága az összes CPU-nak az adott virtuális gépben.	%
cpu.usagemhz.average	Az aktívan használt virtuális CPU értéke. Ez a host által látott CPU használat, és nem a vendég operációs rendszer által megfigyelt.	MHz

7.2. táblázat. Néhány lemezkezeléssel kapcsolatos metrika

Metrika	Leírás	Mértékegység
disk.deviceLatency.average	Egy SCSI lemezművelet végrehajtásának átlagos ideje a fizikai eszköz esetében.	milliszekundum
disk.kernelLatency.average	Az az átlagos idő, amit a VMkernel egy SCSI lemezművelet végrehajtásával tölt.	milliszekundum
disk.totallatency.average	Az az átlagos idő a mérési időablakban, amíg a vendég operációs rendszer által kiadott SCSI lemezművelet végrehajtódik a virtuális gépen. A kernelLatency és a deviceLatency értékének összege.	milliszekundum

7.3. táblázat. Néhány hálózattal kapcsolatos metrika

Metrika	Leírás	Mértékegység
net.transmitted.average	A mérési ablakban elküldött adat átlagos mennyisége. Az érték a hálózat sávszélességét jelenti. VM esetében a virtuális, host esetében a fizikai interfészen keresztül átküldött adat mennyiségét jelenti.	kB/s