Alkalmazás performancia monitoring

téma kutatói összefoglalója

Dr. Huszák Árpád

Hallgató: Koszticza Marcell
Céges konzulens: Sándor Csaba, NISZ Nemzeti Infokommunikációs Szolgáltató Zrt.
Egyetemi konzulens: Dr. Huszák Árpád

Az alkalmazások monitorozására megfelelő eszközök segítségével mélységi betekintést kaphatunk az alkalmazásunk működésébe. A kapcsolódó feladatok leginkább az esetleges hibás működések okainak feltárása, illetve a szolgáltatás minőségének felügyelete. Lehetőségünk nyílik olyan futási jellemzők megfigyelésére, amikből következtethetünk a rendszerünk jövőbeli állapotára, esetleg előre jelezhetjük a hibás működést mielőtt az megtörténne. A munka célja az volt, hogy megvizsgáljuk, miként lehet értékes adatokat kinyerni a DynaTrace nevű alkalmazás monitoring eszközön keresztül a vizsgált rendszerből, illetve miként lehet ezeket az adatokat feldolgozni.

Rendszer felépítés

Az általunk használt monitoring alkalmazás a DynaTrace Appmon 7-es verziója volt, mely képes többféle komponensen keresztül követni az alkalmazás működését. Erősségei közé tartozik a topológiai megjelenítés, illetve a működési folyamatok feltérképezése. Nagy hátrány volt azonban a kinyert adatok tárolásának módja, ugyanis a DynaTrace aggregálva tárolja az adatokat, emiatt nehéz egy-egy konkrét mérési eredményhez való hozzáférés. Tovább nehezíti a helyzetet, hogy az élő nyers adatok kinyerésére sincs egyértelmű módszer, hiszen a Dynatrace alapvetően úgy tervezték, hogy az alkalmazáson belül történjen meg a mért adatok feldolgozása is. Számunkra ez a megoldás nem kielégítő, hiszen az adatok saját algoritmussal történő feldolgozása több lehetőséget kínál, mint a beépített metódusok alkalmazása. Az mérési adatok kinyerésére két lehetőséget találtunk:

1. Reports funkció, mely kevesebb konfigurálást igényel, mivel az éppen megtekintett dashboard adataiból generál egy csv fájlt. Habár a módszerrel elérhetők a régebbi adatok is, és beállítható bizonyos időközönkénti exportálásra, nem tűnt praktikus megoldásnak.
2. DynaTrace Business Transaction, mely megvalósítja a szokásos Performancia Monitoring funkciókat, és lehetőséget kínál a megfigyelt alkalmazásban az adatbányászatra. Ahogy a Business Transaction neve is sugallja első sorban az alkalmazás üzleti jellegű adatainak kinyerésére szolgál, de használható egyéb adatokra is. Maga a Business Transaction egy adat lekérdezés, amivel lekérhetjük a számunkra érdekes adatok halmazát. A lekérdezés úgynevezett measure-ök, amelyek sablonok alapján konfigurálható értékek mentén dolgozik három beállítási típussal: szűrés (beállított értékeket kiszűr), visszaadott érték (megadható, milyen értékeket akarunk megkapni), illetve splitting (amely az SQL groupby funkciójához hasonlít).

A NISZ által DynaTrace-el felügyelt rendszerében (jelen esetben egyik működtetett weblapjuk) felkonfigurálásra került egy Business Transaction, melynek célja a honlapra bejövő Web

Requestek számának megállapítása, illetve a feladó IP-címének a megállapítása. Az elkészült Business Transaction exportálására a DynaTraceben egy beépített eszköz a Real Time Business Transaction Feed nyújt lehetőséget. Az adatokat a Google Protobuf Message formátumban küldi át. Esetünkben a fontosabb adatok: web requestek száma (mivel a rendszer minden egyes web requestre generál egy üzenetet, ezért ez mindig 1), IP-cím (a web request forrása), üzenet információi (Business Transaction neve, szerver neve, időbélyeg, stb.)


```
{"name":"Web Requests - 0s to
ls","application":"easyTravel
portal","systemProfile":"easyTravel","server":"centos-okt",
"type":"PUREPATH","startTime":"2018-02-26
20:17:55.284+0100","failed":false,"responseTime":117.725824
00031388}
```


1. ábra Egy web requesthez tartozó Protobuf messageből generált Json formátumú üzenet

Az üzenetek fogadásához és tárolásához szükségünk van egy eszközre ami, ezt kezeli. A Big Data Business Transaction Bridge egy olyan szolgáltatás a Dynatrace-en belül, ami a Real Time Business Transaction Feed üzeneteit tudja fogadni és különböző módon tárolni. Ehhez az opensource Apache Flume-ot használja illetve kiegészíti ezt egy általa nyújtott jar file-al. Mivel hivatalos támogatás nincs ehhez az eszközhöz, és tesztelve is csak az 1.3.1-es illetve 1.7-es Flume verziókon van, ezért az utóbbi verziót használtuk.

Apache Flume

A Flume az Apache Hadoop projekt része, melynek célja nagy mennyiségű adat megbízható továbbítása a Hadoop Distributed File System felé.

Flume konfigurálása egy virtuális gépen történt meg a NISZ Zrt. hálózatán belül, ugyanis biztonsági megfontolásokból a gyűjtött adatokat nem lehet továbbítani a hálózaton kívülre. A virtuális gépen CentOS 7-et használtunk, ahol egy .properties fájlban lett beállítva a megfelelő konfiguráció, vagyis hogy milyen forrás, nyelő és csatorna beállítások mellett fusson a Flume. Ezek után már csak el kellett indítani, hogy a számára kijelölt porton figyelje az adatstreamet, amit a háttérben futtattunk a logok átirányításával a konzolról egy szöveges fájlba. Miután a Flume már figyelt a virtuális gép megfelelő portján, már csak annyi volt a feladat, hogy beállítsuk a DynaTraceben a Business Transaction lekérdezését, illetve kiküldjük a megfelelő cím ismert portjára. Ezután a Flume a virtuális gép kijelölt könyvtárába írta ki percenként az adatokat egy JSON fájlba.

Az adatgyűjtést április végén sikerült elkezdeni, és egy fél nap kihagyástól eltekintve (a DynaTrace licensz miatt) folyamatosan érkeztek az adatok.

Apache Hadoop

A Hadoop egy olyan környezet, ami a skálázható adattárolás és művelet végzés elérésére törekszik. A Hadoop előtt az alapvető paradigma az volt, hogy kivesszük az adatot az adatbázisból, elvégezzük rajta a számításokat, majd az eredményeket visszatesszük. A Hadoop ezen annyit változtatott, hogy a műveletvégzést viszi az adathoz, vagyis maga a Hadoop nemcsak adatokat tárol, hanem műveleteket is képes végezni a tárolt adatokon.

A Hadoop parancsok egészen hasonlóan működnek, mint az alap Linux filesystem ugyanolyan parancsai, nyilvánvaló különbség, hogy a Hadoop nem látja a saját könyvtár, illetve a szülőkönyvtár bejegyzését, illetve a jogok mellet a számok a Hadoop esetében a replikációs faktor, ami megmutatja, hogy a Hadoop hány példányban tárolja az adott fájlt.

```
marcell@debian:~/python_workdir$ hdfs dfs -ls
Found 4 items
drwxr-xr-x   - marcell marcell       4096 2018-05-30 01:34 dataset_for_ml
-rw-r--r--   1 marcell marcell       3193 2018-05-30 02:09 padas_df
drwxr-xr--   - marcell marcell       4096 2018-05-30 05:59 pp
-rw-r--r--   1 marcell marcell     634010 2018-05-30 06:07 traffic.csv
marcell@debian:~/python_workdir$
```

2. ábra A HDFS és a Linux Filesystem ls parancsai

Úgy döntöttem, hogy bár teljesen elosztott rendszerként nem tudom használni, mivel nem állt számomra rendelkezésre további erőforrás, illetve mivel az az egy virtuális gépemnek sem voltak túl erősek az erőforrásai, hogy a local módban fogom futtatni a Hadoopot. Local Mode (Standalone) esetén a Hadoop egy komponensen fut és ekkor nem használhatók az elosztott rendszerre tervezett funkciók.
Mivel a Flume tud önállóan működni Hadoop nélkül, ezért nincs feltétlenül szükség rá. A rendszer tervezésekor még nem tudtuk, hogy mekkora adatmennyiség fog a munka végére a rendelkezésemre állni, ezért fent akartuk tartani a lehetőséget arra, hogy akár a közeljövőben, ha bővítésre van szükség az egyszerűen megoldható legyen. A rendszertervezés későbbi fázisában jött be még egy szempont az adatfeldolgozással kapcsolatba, aminek hatására, mégis megtartottuk a Hadoopot, mivel az adatfeldolgozásra az Apache Sparkot választottuk, aminek szüksége van a Hadoop könyvtáraira.

Apache Spark

Miután az adatgyűjtés, illetve ennek tárolása megoldódott, az értékes adatok kinyerése volt a következő feladat. Célunk az volt, hogy a requestenként bejövő üzenetek adataiból előállítsunk egy olyan időalapú sorozatot, ami megmutatja, hogy adott intervallumon belül mennyi web request érkezett be a honlapra. Erre a feladatra három megoldást láttam:

- MapReduce: mivel Hadooppal dolgoztunk, felvetődött, hogy használjuk annak a beépített adatfeldolgozó keretrendszerét a MapReduce-t. Ennek meglett volna az az előnye, hogy nem kellett volna mozgatni az adatokat, illetve minden készen állt az alkalmazására. A Hadoop ökoszisztémában a MapReduce-nál már létezik jobb adatfeldolgozó keretrendszer: az Apache Spark
- Az adatok kimásolása másik számítógépre: a megoldás problémája, hogy az adatok bizalmasak, ezért nagyfokú körültekintéssel kellett volna kezelnem. Másrészt még mindig nem tudtuk pontosan az adatok mennyiségét, vagyis, hogy mennyi adaton kell majd számításokat végeznem, és hogy bele fognak-e férni a memóriába.
- Spark: úgy tartottuk, hogy érdemes lenne a virtuális gépen elvégezni az adatfeldolgozást, és erre a Spark egy jó eszköznek ígérkezett ezért emellett döntöttünk. Az Apache Spark egy adatfeldolgozó keretrendszer Hadoopra, melynek előnye, hogy elvégzett művelet között az adatokat a memóriában tárolja, használható Pythonnal, R-rel és Javával, konzolos felületen is használható, beépített Machine learning library, Streaming adatkezelés, gráf feldolgozás, SQL.

Adatok átalakítása

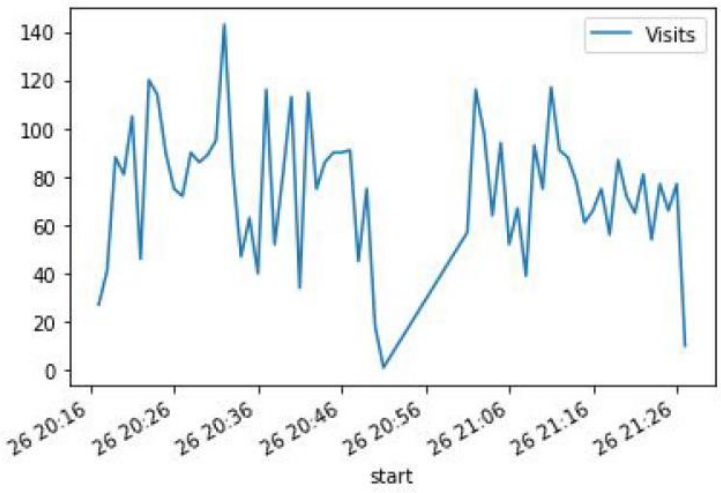
Szerencsére a Spark keretrendszer használható Pythonnal is, ezért mi is ezt használtuk a további adatfeldolgozáshoz. Mivel a Spark strulturált adatokat szolgáltat, azokat könnyedén lehetett egy DataFramebe beolvasni. Az adatokat a Pyspark automatikusan generálja a megfelelő formátumba.

Szűrőt beállításával csak az általunk definiált Business Transaction értékeit veszi figyelembe, majd kiválasztjuk a Web Request kezdeti idejét, és ezt átcastoljuk Timestamp formátumúra, ezek után pedig kiválasztjuk a maradék számunkra fontos oszlopokat. Az adatok megjelenítésével, illetve a séma kiírásával tudjuk ellenőrizni a művelet eredményességét. Következő lépés, hogy megszámoljuk egy bizonyos időszakon belül a web requestek számát. Ehhez bevezettünk egy segédváltozó oszlopot, ami minden sorhoz generált egy 1-es értéket. A beépített window függvény segítségével 1 perces intervallumokra osztottuk fel a startTimeTS oszlopot, majd ezeken az intervallumon aggregáltuk a segédváltozó értékét. Itt érdemes megjegyezni a Spark egyik sajátosságát: ugyan a Spark engedélyezi a felhasználó által írt függvények (UDF-User Defined Functions) alkalmazását, ez alapvetően kerülendő, ugyanis a Spark a JVM-ben fut, és minden ilyen függvénynél, először ki kell venni a JVM-ből az adatokat, majd a számítás végeztével vissza kell rakni őket oda, ezzel jelentős overheadet okozva. A DataFrame kiírásával a képernyőre meggyőződhetünk arról, hogy megfelelően végezte el a műveleteket. Ez az adat már nagyjából megfelel arra, hogy predikciót építsünk rá. Kisebb problémát jelentett azonban a Timewindow oszlop értékei, mivel ezek beágyazott tupleként voltak benne a DataFrame-ben, ezért ezek alapján egy új DataFrame-t készült, ahol külön oszlopokként kezelhetjük azokat.

```
+-----------------------------------------+----------------------+
|Timewindow                               |Request_count_by_minute|
+-----------------------------------------+----------------------+
|[2018-02-26 20:17:00, 2018-02-26 20:18:00]|27                    |
|[2018-02-26 20:18:00, 2018-02-26 20:19:00]|41                    |
|[2018-02-26 20:19:00, 2018-02-26 20:20:00]|88                    |
|[2018-02-26 20:20:00, 2018-02-26 20:21:00]|81                    |
|[2018-02-26 20:21:00, 2018-02-26 20:22:00]|105                   |
```

3. ábra A window_df DataFrame első 5 eleme

Az éles rendszerben a Business Transaction Bridge által generált összes üzenet mérete elérte a 2 GB-ot. Az adatok szűrése és jobb formátumba történő kiírása után azonban már csak pár MB lett. Itt úgy döntöttünk, hogy ilyen méreteknél már érdemesebb lehet kimásolni a virtuális gépről az adatokat, és áttérni a Pandas könyvtár használatára (a Pandas egy adatelemzésre használt könyvtár Pythonhoz).

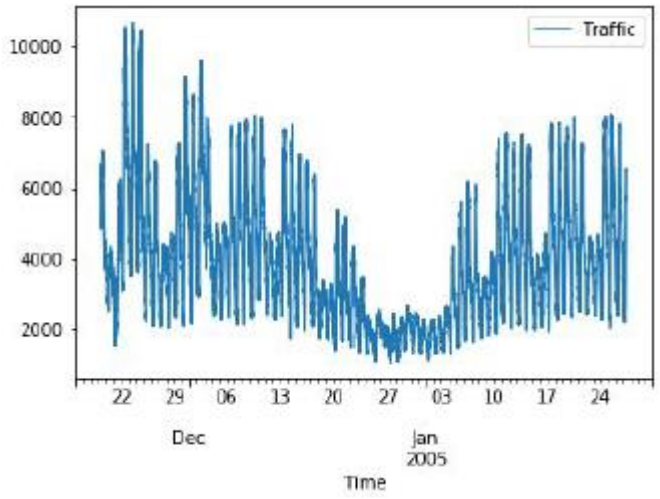


4. ábra A demókörnyezetben generált web requestek az idő függvényében

Az ábárából nem derül ez ki, de valójában a grafikon közepén nem jöttek web requestek a rendszerre. Elképzelhető, hogy egy rendszerre nem érkeznek kérések egy adott intervallumban, szóval az adott intevallumban a Visits oszlop elvárt értéke 0, de nálunk ilyen intervallumok egyáltalán nincsenek is, ezért generálnunk kell őket. Ezt úgy érhetjük el, hogy készítünk egy DataFrame-t egy oszloppal, ami a kívánt időintervallumot fedi le a megfelelő frekvenciával. Ezután az új és a már meglévő DataFrame-t egyesítjük egy outer joint művelettel. Ekkor azokhoz az időbélyegekhez, amikhez nem volt érték az eredeti DataFrame-ben egy NaN érték fog kerülni, ezt egy függvénnyel kicseréljük 0-kal.

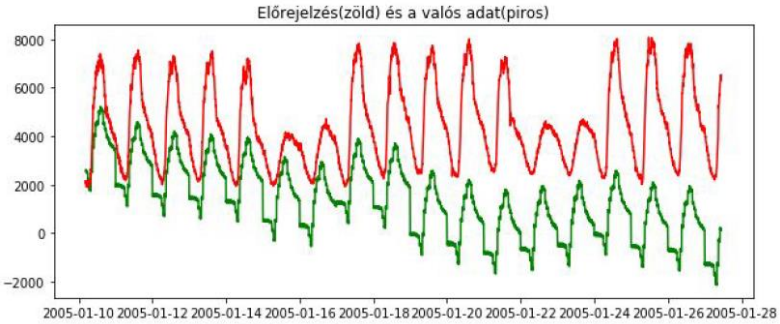
Prediktív modell lineáris regresszióval

A felhasznált adathalmaz nem elég nagy mennyiségű ahhoz, hogy predikciós modellt építsünk rá, ezért felkészülésként a következő féléves feladatra egy publikus datasetet alkalmaztunk, ami elég hasonló az adatainkhoz. A felhasznált dataset egy ISP hálózatán mért adatforgalom bitekben, 5 perces intervallumokban.



5. ábra A használt publikus adathalmaz ábrázolva

Miután kettébontjuk az adathalmazunk 1:3-hoz arányban egy test és egy train részre, a train adatokkal optimalizáljuk a becslőegyenlet együtthatóit a testtel pedig ellenőrizzük a működését. A modellnek megadjuk a bemenő változókat (idő és ebből származtatott számok), illetve a megbecsülni kívánt érték train részét (adatforgalom bitekben).



6. ábra Az alkalmazott predikciós modell becslése (zöld), illetve a valós eredmények (piros)

A becslés rosszul sikerült, aminek oka a modell és az adathalmaz: Mivel az adathalmazban novembertől decemberig folyamatos csökkenés figyelhető meg, ezért a modell feltételezi, hogy továbbra is csökkenő a tendencia, még akkor is, ha a január eleji adatok már növekvő tendenciát mutatnak. Ettől függetlenül a napi változásokat egész jól becsli, mivel az elég hasonló formát szokott felvenni.

Eredmények, értékelés

Munkánk során a Dynatrace, Apache Floom, Hadoop, Spark megoldások alkalmazások segítségével sikerült kinyerni adatokat az éles környezetből, melyeket már fel lehet dolgozni vagy tárolni. A folytatásban a megismert módszereket újabb, az eddiginél komplexebb feladatkörben szeretném alkalmazni. Nehézséget jelentett, hogy viszonylag későn (április vége) tudtuk elkezdeni érdemi munkát végezni, ugyanis ekkortól állt rendelkezésre a virtuális gép.