Hadoop, Hive a Spark: Využití pro data mining

Václav Zeman, KIZI VŠE

VŠE: Katedra informačního a znalostního inženýrství

- Hlavní činnosti:
  - Data Mining a Machine Learning
    - Nástroje: LispMiner a **EasyMiner**
  - Distribuované výpočty
    - S využitím prostředků **CESNET Metacentra**
  - Linked data a sémantický web
    - Česká DBpedia
- Spolupráce s FIT ČVUT a MFF UK
  - EasyMiner, OpenBudgets.eu, LinkedTV
- Projekty podporované **Fondem rozvoje**

EasyMiner
easy association rule mining

OpenBudgets.eu

LINKEDTV

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FIT

MetaCentrum

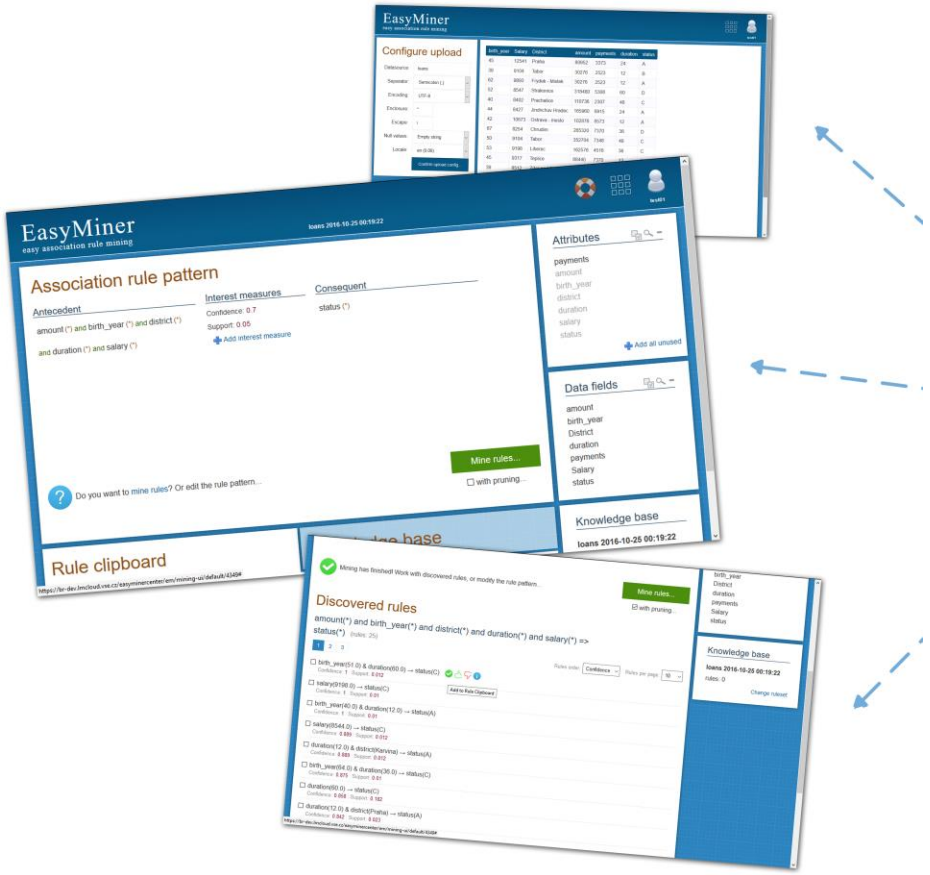
VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

VŠE

DBpedia

EasyMiner

- Akademický nástroj pro:
  - Hledání pravidel v datech
  - Tvorbu klasifikačních modelů z pravidel
  - Detekci anomálií
- Do roku 2014:
  - Postaveno na nástroji LispMiner
- Nynější verze používá:
  - R (arules, rCBA, fpmoutliers)
  - Hadoop, Hive a Spark



Hledání pravidel

- Základní úloha v oblasti dolování dat
- Databáze transakcí
- Počet pravidel = $\sum_{k=2}^{N} \binom{N}{k} \cdot (2^k - 2)$
  - Složitost narůstá exponenciálně s počtem položek v databázi *(kombinatorický problém)*
- Prořezávací algoritmy:
  - Apriori, FP-growth, Eclat, LCM…
- Paralelní verze:
  - PFP: parallel FP-growth (implementováno ve Sparku)

Příklad: nákupní košík

Databáze transakcí

1: {milk, bread, butter} N = 3
2: {bread}
3: {milk, break}

Nalezená pravidla

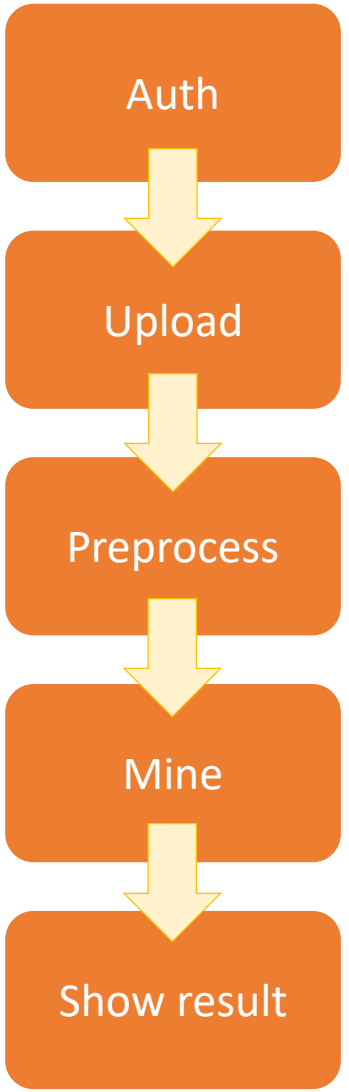
{milk} => {bread} : supp=0.66, conf=1
{break} => {milk} : supp=0.66, conf=0.66
{milk, break} => {butter} : supp=0.33, conf=0.5
…

Projekt: CESNET - č. 540/2014

- Podporováno fondem rozvoje
- Dolování pravidel a tvorba klasifikátorů nad většími daty
  - > 500MB
  - Distribuovaný výpočet
  - Memory+Disk based přístup
- Technologie: Hadoop, Hive a Spark
- Nasazeno na Hadoop cluster **Metacentra**
- Ovládání skrze grafické rozhraní
  - Na pozadí prováděny MapReduce úlohy a Spark joby
  - Dávkový přístup, YARN plánovač

Hledání pravidel na clusteru

1. Autentizace přes Kerberos
2. Nahrání datasetu na cluster
3. Předzpracování a transformace dat na clusteru
4. Distribuované hledání pravidel
5. Distribuované prořezávání pravidel a tvorba klasifikačních modelů
6. Čtení a interpretace výsledků



Hadoop klient

- Instalace na Debian:
  - apt-get install hadoop-client hive spark-python
  - Repozitář: http://scientific.zcu.cz/repos/hadoop/cdh5/debian/...
- Konzole: hadoop, yarn, hive (beeline), spark
- Přístup z kódu – JVM (Java, **Scala**, Clojure…)
  - Hadoop: Maven ("org.apache.hadoop" % "hadoop-client")
  - Hive: JDBC ("org.apache.hive" % "hive-jdbc")
  - Spark: SparkLauncher ("org.apache.spark" %% "spark-launcher")

Přihlášení na Hadoop cluster Metacentra

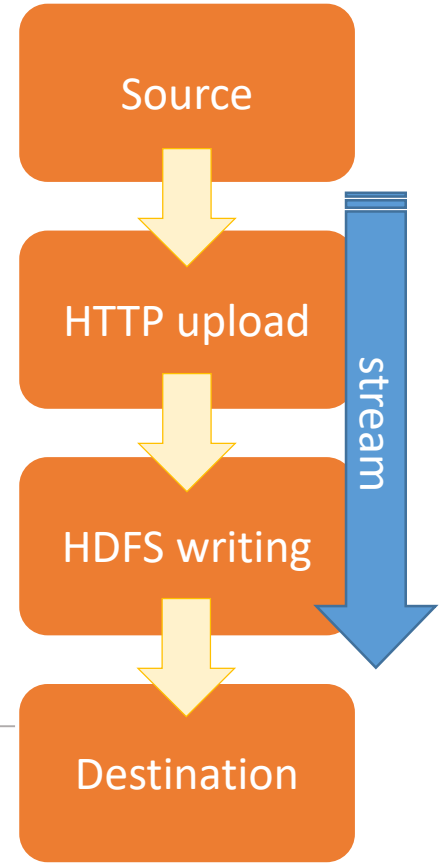
- Skrze Kerberos
  1. Tvorba Kerberos lístku
  2. Použití lístku k přihlášení
  3. Obnova lístku
- Řešení v EasyMineru
  1. Vygenerování **keytab** souboru
  2. Cron job pro tvorbu Kerberos lístků
     - kinit -k -t easyminer.keytab easyminer@META
  3. Přihlášení z Java API

Environment variables
KRB5_CONFIG=/path/to/krb5.conf
KRB5CCNAME=FILE:/path/to/kerberos/ticket
HADOOP_USER_NAME=<username>

```
UserGroupInformation.setLoginUser(UserGroupInformation.getBestUGI("/path/to/kerberos/ticket", "<username>"))
UserGroupInformation.getLoginUser.reloginFromTicketCache()
```

Nahrání datasetu na Hadoop cluster

- Skrze konzoli:
  - hadoop fs –put <local://source> <hdfs://destination>
- Řešení v EasyMineru
  - Skrze Java API
  - Nahrávání přes Java OutputStream




```
import org.apache.hadoop.fs.FileSystem
import org.apache.hadoop.conf.Configuration
…
Configuration conf = new Configuration()
conf.addResource(“<hadoop-config-file>”)   //core-site.xml,hdfs-site.xml,yarn-site.xml,mapred-site.xml
…
Writer writer = FileSystem.get(conf).create(“/path/to/hdfs/destination/file”)
writer.write(<byte array>)
…
```

Transformace dat na clusteru

- Použití technologie **Apache Hive**
- SQL dotazy nad tabulkou v HDFS (csv, orc, parquet, avro)
  - Automaticky převedeny na MapReduce úlohy
  - Alternativy: SparkSQL, Drill, Impala…
- Vzdáleně skrze JDBC:
  - HiveServer2: zpracovává vzdálené SQL dotazy

```
import java.sql.DriverManager
…
Class.forName("org.apache.hive.jdbc.HiveDriver")
String driver = “jdbc:hive2://hador-c1.ics.muni.cz:10000/<database>;principal=hive/hador-c1.ics.muni.cz@ICS.MUNI.CZ”
Connection conn = DriverManager.getConnection(driver, “<username>”, “”)
conn.prepareStatement(“INSERT INTO TABLE table1 SELECT value, COUNT(id) FROM table0 GROUP BY value”).execute()
```

Transformace dat na clusteru

- Řešení v EasyMineru:
  - Data ukládána do HDFS jako CSV
  - Transformace skrze Hive do formátu ORC
- EAV antipattern (entity-attribute-value)
  - Rychlejší přidávání a odstraňování sloupců
  - Není nutné joinovat tabulky během transformace
  - Vhodné pro řídké tabulky
  - Lepší škálovatelnost: PARTITIONED BY attribute

ID	Col1	Col2	Col3
1	1	2	2
2	0	3	NA
3	2	NA	1

to EAV model

ID	Attribute	Value
1	1	1
1	2	2
1	3	2
2	1	0
2	2	3
3	1	2
3	3	1

PARTITIONED BY attribute

Attribute 1

ID	Value
1	1
2	0
3	2

Attribute 2

ID	Value
1	2
2	3

Attribute 3

ID	Value
1	2
3	1

Distribuované dolování pravidel

- Apache Spark
  - MLlib: PFP - Parallel FP-growth
- Cluster mód v Metacentru: Spark on YARN
- Řešení v EasyMineru:
  - V HDFS uloženo:
    1. Spark dolovací aplikace „spark-miner.jar“
    2. Kompletní Spark (pro YARN) „spark-assembly.jar“
    3. Pomocné knihovny
    4. Zadání úlohy
  - Spuštění Spark aplikace přes SparkLauncher

Environment variables
SPARK_HOME=/usr/lib/spark

Distribuované dolování pravidel

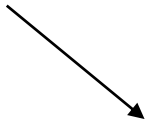
- Spuštění Spark aplikace přes SparkLauncher
  - YARN cluster mode
  - Driver/Executors = YARN containers

EasyMiner mining launcher

```
new SparkLauncher()
.setAppResource(“hdfs://<spark-miner.jar>”)
.setMainClass(“eu.easyminer.sparkminer.MinerLauncher”)
.addAppArgs(“hdfs://<input-task.xml>”)
.setMaster("yarn")
.setDeployMode("cluster")
.addFile(“local://path/to/hive-site.xml”)
.addFile(“local://path/to/log4j.properties”)
.setConf("spark.yarn.jar", “hdfs://<spark-assembly.jar>”)
.addJar(“hdfs://<other-required-library.jar>”)
.launch()
```

Spark configs

```
spark.executor.cores=4
spark.executor.instances=16
spark.executor.memory=8g
```



Max: 64 jobs

Stages for All Jobs

Completed Stages: 24

Completed Stages (24)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input
49	count at CBAM2.java:155	+details	2017/03/29 11:55:12	30 ms	12/12	7.6 GB
47	count at CBAM2.java:155	+details	2017/03/29 11:55:12	35 ms	12/12	7.6 GB
45	count at CBAM2.java:151	+details	2017/03/29 11:55:11	0.2 s	12/12	7.6 GB
43	count at CBAM2.java:92	+details	2017/03/29 11:55:11	34 ms	12/12	7.6 GB
41	collectAsMap at CBAM2.java:90	+details	2017/03/29 11:55:11	68 ms	12/12	
40	mapToPair at CBAM2.java:87	+details	2017/03/29 11:55:11	0.4 s	12/12	7.6 GB
38	reduce at CBAM2.java:258	+details	2017/03/29 11:53:22	1.8 min	12/12	7.6 GB
36	zipWithIndex at CBAM2.java:193	+details	2017/03/29 11:53:17	4 s	11/11	6.5 GB
34	takeOrdered at FPGrowthMiner.java:300	+details	2017/03/29 11:53:17	0.1 s	12/12	832.0 KB
31	count at FPGrowthMiner.java:95	+details	2017/03/29 11:53:17	44 ms	12/12	832.0 KB
28	count at FPGrowthMiner.java:91	+details	2017/03/29 11:53:17	44 ms	12/12	832.0 KB
25	count at FPGrowthMiner.java:87	+details	2017/03/29 11:53:17	78 ms	12/12	832.0 KB
22	count at FPGrowthMiner.java:83	+details	2017/03/29 11:53:17	0.1 s	12/12	1381.8 KB
19	collect at FPGrowthMiner.java:134	+details	2017/03/29 11:53:17	69 ms	12/12	1381.8 KB
16	count at FPGrowthMiner.java:132	+details	2017/03/29 11:53:16	0.2 s	12/12	
13	count at FPGrowthMiner.java:81	+details	2017/03/29 11:53:16	42 ms	12/12	7.1 GB
11	count at FPGrowthMiner.java:79	+details	2017/03/29 11:53:16	0.3 s	12/12	
10	flatMap at FPGrowth.scala:168	+details	2017/03/29 11:53:12	4 s	12/12	7.1 GB
8	collect at FPGrowth.scala:149	+details	2017/03/29 11:53:11	0.9 s	12/12	
7	map at FPGrowth.scala:146	+details	2017/03/29 11:53:09	3 s	12/12	7.1 GB
5	count at FPGrowth.scala:114	+details	2017/03/29 11:53:08	0.3 s	12/12	7.1 GB
3	count at FPGrowthMiner.java:74	+details	2017/03/29 11:53:08	0.1 s	12/12	7.1 GB
1	count at FPGrowthMiner.java:69	+details	2017/03/29 11:52:32	36 s	12/12	
0	groupBy at TransactionsFrameReader.java:184	+details	2017/03/29 11:52:17	15 s	12/12	3.7 MB

Prořezávání

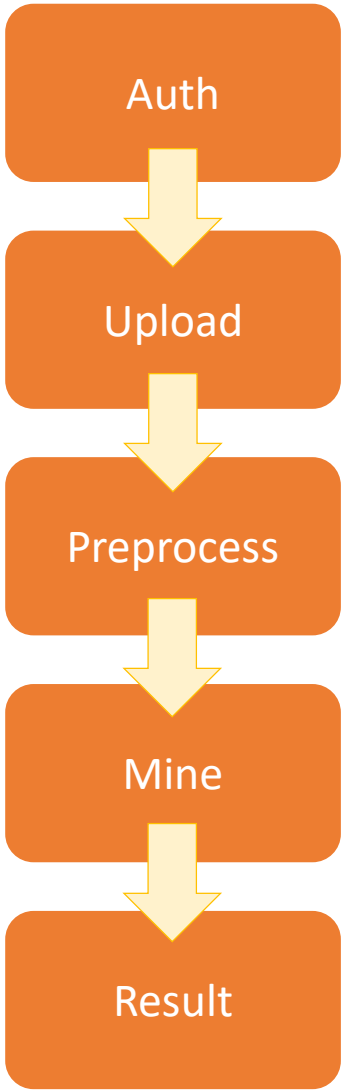
Generování pravidel

MLlib: FP-growth

Příprava dat

Operace na clusteru

1. Ukládání datového zdroje
   - HDFS stream + Hive
   - 2x MapReduce
2. Předzpracování datového zdroje
   - Hive
   - 3x MapReduce
3. Dolování asociačních pravidel
   - Spark
   - 13x Spark Jobs
4. Prořezávání pravidel (tvorba klasifikátoru)
   - Spark
   - 7x Spark Jobs



Sledování stavu Spark úlohy

- Z konzole
  - yarn logs -applicationId <application-id>
  - yarn application –kill <application-id> *(zabití úlohy)*
- Přes prohlížeč
  - https://hador-c2.ics.muni.cz:8090/cluster
  - Nutná autentizace přes Kerberos
- Z Java kódu:

SparkLauncher handler and listeners

```
SparkAppHandle sah = new SparkLauncher()
.startApplication(new MyListener())
sah.getAppId()
sah.kill()
```

YARN monitoring

```
YarnClient yarn = YarnClient.createYarnClient()
yarn.init(conf)
yarn.start()
Process process = new SparkLauncher().launch()
Reader reader = new InputStreamReader(process.getErrorStream)
…
yarn.getApplicationReport(“<application-id>”).getYarnApplicationState
yarn.killApplication(“<application-id>”)
```

Ukládání výsledků dolování

- Výsledky uloženy do souboru v HDFS
- Čtení souboru z HDFS

```
import org.apache.hadoop.fs.FileSystem
InputStream is = FileSystem.get(conf).open("/path/to/hdfs/source/file")
is.read(bytes)
```

- Smazání souboru

```
import org.apache.hadoop.fs.FileSystem
FileSystem.get(conf).delete("/path/to/hdfs/source/file", false)
```

Interpretace výsledků

- district(Uherske Hradiste) → rating(C)
  Confidence: 0.889 | Support: 0.012
- salary(8544) → rating(C)
  Confidence: 0.889 | Support: 0.012
- salary(9198) & district(Liberec) → rating(C)
  Confidence: 1 | Support: 0.01
- salary(8544) & district(Uherske Hradiste) → rating(C)
  Confidence: 0.889 | Support: 0.012
- age(51) & payments(500) → rating(C)
  Confidence: 1 | Support: 0.01
- age(49) & payments(500) → rating(C)
  Confidence: 1 | Support: 0.01
- age(60) & payments(2500) → rating(A)
  Confidence: 1 | Support: 0.01

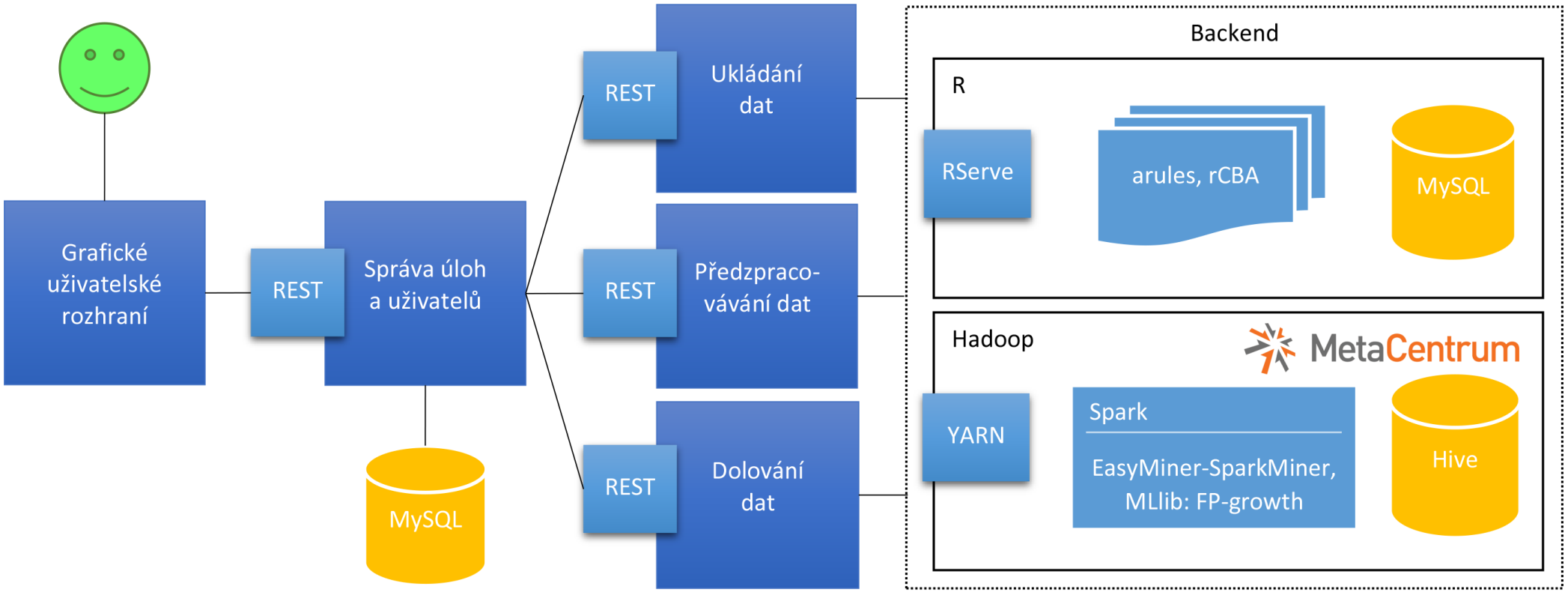

```
▼<AssociationRule id="6511" antecedent="cdnt_37" consequent="cdnt_32">
   <Text>district(Pardubice) → rating(C)</Text>
   <IMValue name="BASE" type="%All">0.0116486</IMValue>
   <IMValue name="CONF" type="%All">0.8</IMValue>
   <FourFtTable a="72" b="18" c="3555" d="2536"/>
 </AssociationRule>
▼<AssociationRule id="6512" antecedent="cdnt_38" consequent="cdnt_32">
   <Text>age(25) → rating(C)</Text>
   <IMValue name="BASE" type="%All">0.0116486</IMValue>
   <IMValue name="CONF" type="%All">0.8</IMValue>
   <FourFtTable a="72" b="18" c="3555" d="2536"/>
 </AssociationRule>
▼<AssociationRule id="6513" antecedent="cdnt_39" consequent="cdnt_32">
   <Text>age(47) → rating(C)</Text>
   <IMValue name="BASE" type="%All">0.020385</IMValue>
   <IMValue name="CONF" type="%All">0.823529</IMValue>
   <FourFtTable a="126" b="27" c="3501" d="2527"/>
 </AssociationRule>
```


Rule details

district(Uherske Hradiste) → rating(C)

	consequent	¬ consequent
antecedent	72	9
¬ antecedent	3555	2545

Architektura nástroje EasyMiner



Závěr

- Vývojová verze nástroje EasyMiner na:
  - https://easyminer-demo.lmcloud.vse.cz
- Dva backendy:
  - R (real-time, in-memory, do 100MB)
  - Hadoop, Hive a Spark (dávkové zpracování, scheduler, více než 100MB)
- Future work:
  - Rozšíření uživatelského rozhraní
  - Dolování pravidel nad propojenými daty
  - Pokročilé možnosti předzpracování (diskretizační algoritmy)
  - Detekce anomálií - pro malá (v R) i velká data (Hadoop + Spark)

Otázky?