

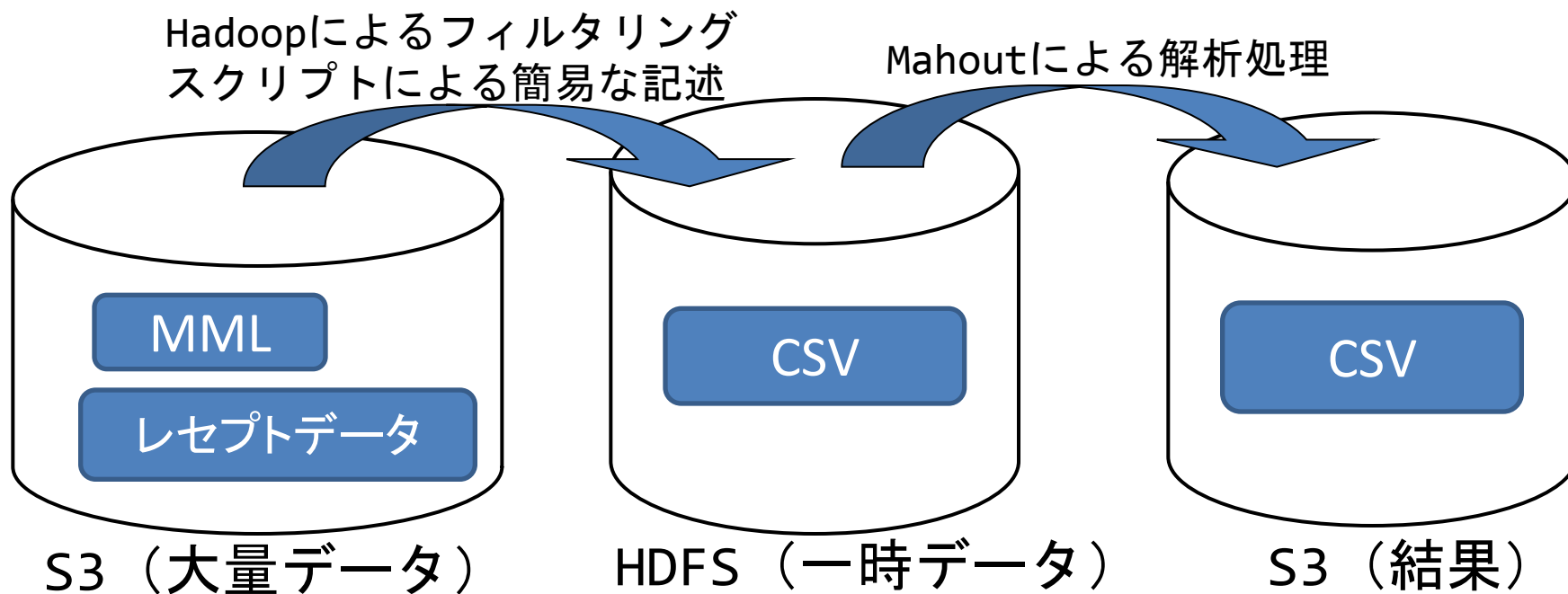
# Mahoutを使った レコメンデーションのプロトタイプ

2011/10/21

アドソル日進株式会社

荒本道隆

- 蓄積されたデータを使って、Mahoutで解析
- まずは、お医者さんが利用することを想定
  - 患者側が利用することはできないだろうか？



- 機械学習・データマイニングのライブラリ
- Java & オープンソース
  - Apache2.0ライセンス
- Hadoop上で動作
  - hadoop-coreが含まれているので、単体でも動作
- AmazonEMR上でも動作
  - MahoutDriver を指定して実行

- Clustering : クラスタリング : 似ているものをグループ化する
- Classification : 分類 : 分類・判別を行う
- Pattern Mining : パターンマイニング : 頻出パターンを抽出する
- Recommender : 推薦 : 興味に合うものを推薦する
- Regression : 回帰 : 数値予測を行う
- Dimension Reduction : 次元縮約 : 説明変数を重要な変数へ縮約
- Evolutionary Algorithms : 進化的アルゴリズム

参考URL : <http://sites.google.com/site/mahoutjp/>

- さらに詳細な情報

– <https://cwiki.apache.org/MAHOUT/algorithms.html>

- ローカルHadoopのスタンドアロンモードで開発
  - Linuxがあれば、hadoopのファイルを展開するだけ
- Amazon EMR (Elastic MapReduce) + S3で実行
  - コマンド一発でHadoop実行環境が準備できる
  - S3の使い勝手も検証
- Streamingモードをできるだけ使用
  - フィルタリングをRubyで記述
    - Rubyは**初心者**です
      - Rubyらしくないコードでスイマセン

# まずはMMLを AmazonEMRで利用

- S3上に大量のMMLが保存されていると仮定
  - 特定の範囲のMMLだけ処理したい
  - MMLの中から特定の要素を抽出
  - 結果をS3上に保存
- 
- まずはAmazonEMRに慣れる&性能測定

S3



結果

AmazonEMR

Rubyによる特定要素抽出

HDFS



# Rubyによる特定要素抽出(xml)

```
#!/usr/bin/env ruby
require "rexml/document"
include REXML
```

```
string = ""
while line = STDIN.gets
  string += line
end
doc = Document.new string
# print doc
```

```
birthday = XPath.first( doc, "//mmlPi:PatientModule/mmlPi:birthday" )
print birthday.text.split("-")[0] + "¥t"
```

```
bodyWeight = XPath.first( doc, "//mmlFcl:birthInfo/mmlFcl:bodyWeight" )
print bodyWeight.text + ","
bodyHeight = XPath.first( doc, "//mmlFcl:birthInfo/mmlFcl:bodyHeight" )
print bodyHeight.text
```

- Rubyによる特定要素抽出

- － ローカルHadoop

```
hadoop jar $HADOOP_HOME/contrib/streaming/hadoop-0.20.2-streaming.jar ¥  
-mapper "ruby mapper1.rb" ¥  
-input ./input/*/*/ ¥  
-output ./output1
```

- － AmazonEMR

```
elastic-mapreduce -j $JOB_ID ¥  
--stream --step-name "mapper1.rb" ¥  
--mapper s3://bucket名/mml/mapper1.rb ¥  
--reducer s3://bucket名/mml/red_dummy.rb ¥  
--input s3://bucket名/mml/input/*/*/ ¥  
--output s3://bucket名/mml/output1
```

# やってみて気付いた事 - 1

- Javaによる実装より簡単だった
- AmazonEMRは「Reduceなし」だとエラーになる
  - 何もしない red\_dummy.rb を作成
- inputにワイルドカードを指定できる
  - s3://bucket名/mml/input/2011/09/
  - s3://bucket名/mml/input/2011/\*/
  - s3://bucket名/mml/input/\*/09/
  - s3://bucket名/mml/input/\*/\*/
  - S3上へデータを格納
    - 「hadoop distcp」を使用

- ファイル数 = map数
  - インスタンス数を増やせば、単純計算で時間短縮
    - AmazonEMRのsmallインスタンスで測定
    - データはS3に格納
  - 100ファイル (4,517,102Byte)
    - 1インスタンス: 16分
    - 1+1インスタンス: 10分
    - 1+5インスタンス: 3分
    - 1+10インスタンス: 2分
  - 1,000ファイル (45,166,848Byte)
    - 1インスタンス: 227分
    - 1+1インスタンス: 100分
    - 1+5インスタンス: 23分
    - 1+10インスタンス: 12分

- AmazonEMRのsmallインスタンスで測定
  - データはS3に格納
  - 100ファイル(4,517,102Byte)
    - 1インスタンス:16分
    - 1+1インスタンス:10分
    - 1+5インスタンス:3分
    - 1+10インスタンス:2分
    - EMR上でHadoop、データはS3:9分
    - EMR上でHadoop、データはローカル:9分
    - ローカルHadoop(PC使用)、データはローカル:2分
    - ローカルシェル(PC使用)、データはローカル:2分

- AmazonEMRのsmallインスタンスで測定
  - データはS3に格納
  - 1,000ファイル(45,166,848Byte)
    - 1インスタンス:227分
    - 1+1インスタンス: 100分
    - 1+5インスタンス:23分
    - 1+10インスタンス:12分
    - EMR上でHadoop、データはS3:84分
    - EMR上でHadoop、データはローカル:77分
    - ローカルHadoop(PC使用)、データはローカル:19分
    - ローカルシェル(PC使用)、データはローカル:17分

- AmazonEMRのsmallインスタンス(0.115\$/時)
  - 参考値 – 2の数値を100倍にして計算
  - 100,000ファイル(4,516,684,800Byte)の予想値
    - 1インスタンス:22700分(379時間)
      - $0.115 \times 1 \times 379 = 43.585\$$
    - 1+1インスタンス:10000分(167時間)
      - $0.115 \times 2 \times 167 = 38.41\$$
    - 1+5インスタンス:2300分(39時間)
      - $0.115 \times 6 \times 39 = 26.91\$$
    - 1+10インスタンス:1200分(20時間)
      - $0.115 \times 11 \times 20 = 25.3\$$

- AmazonEMRのsmallインスタンスで測定
  - Ruby+DOM+XPath → grep+sed+awkでSAXもどき
  - 1,000ファイル(45,166,848Byte)
    - 1インスタンス:227分 → 83分
    - 1+1インスタンス:100分 → 51分
    - 1+5インスタンス:23分 → 10分
    - 1+10インスタンス:12分 → 8分
    - EMR上でHadoop、データはS3:84分 → 50分
    - EMR上でHadoop、データはローカル:77分 → 50分
    - ローカルHadoop(PC使用)、データはローカル:19分 → 5分
    - ローカルシェル(PC使用)、データはローカル:17分 → 0.5分



# レセプトデータを使った レコメンデーション

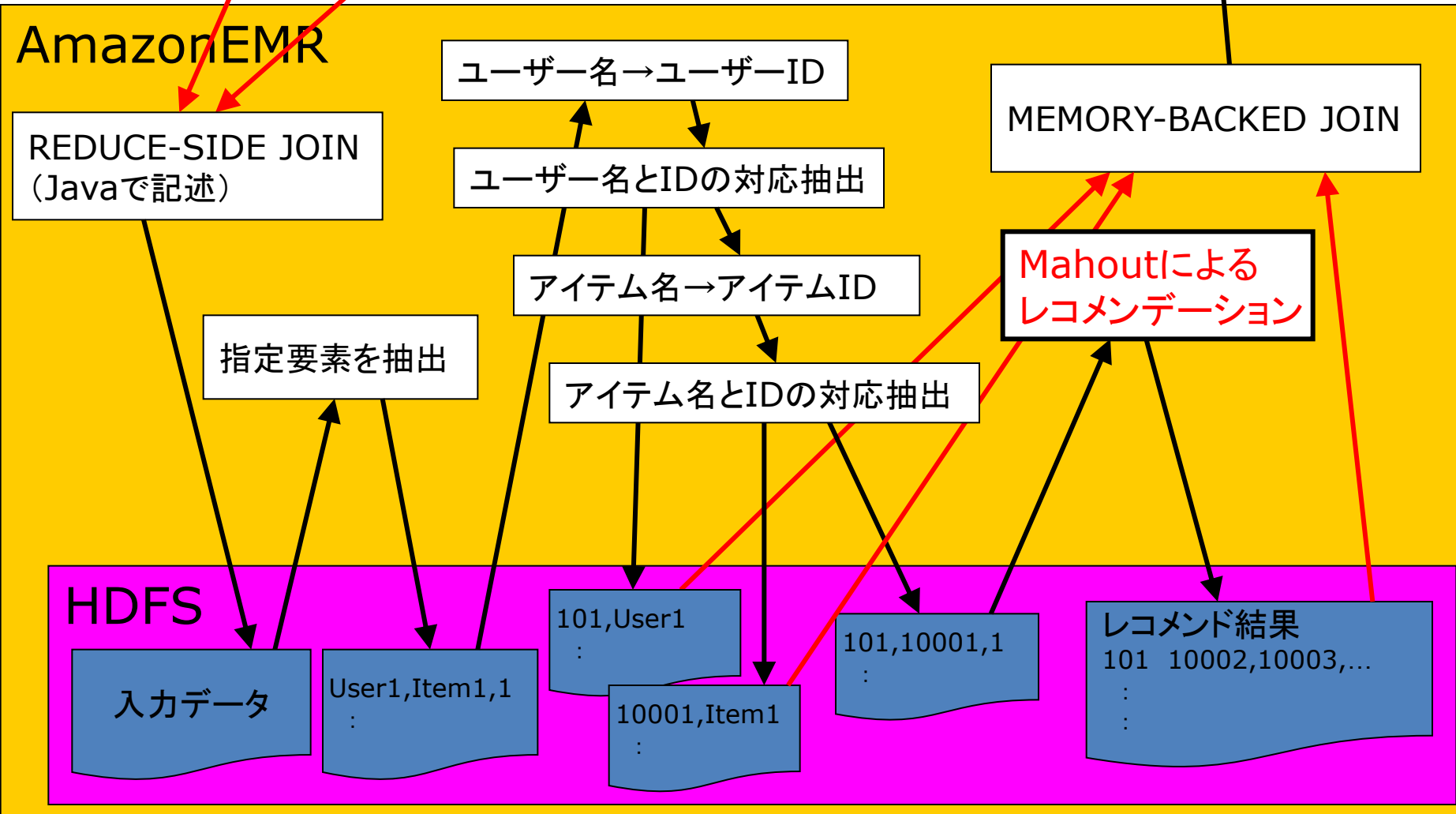
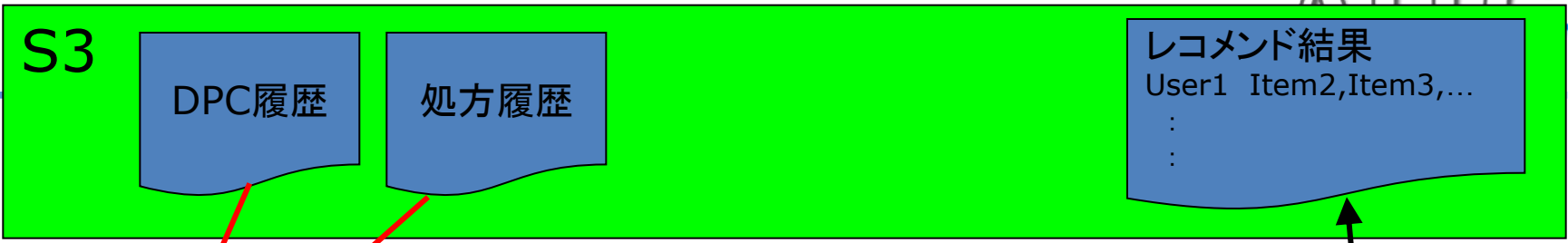
- S3上にレセプトデータがあると仮定
  - DPC履歴ファイル
  - 処方履歴ファイル
  - などなど
- Mahoutによるレコメンデーションを実行
- 結果をS3に保存

- DPC履歴ファイル
  - 施設CD,患者ID,入院年月日,退院年月日,分類番号,....
- 処方履歴ファイル
  - 患者ID, 医師ID,処方年月日,医薬品コード,...
- この2ファイルを患者IDでJOIN
  - JOIN結果を入力データとする

- Mahoutのレコメンデーションの入力フォーマット
  - ユーザーID(long), アイテムID(long), 評価値(double)
    - 患者ID→ユーザーID
    - 医薬品コード→アイテムID
    - 評価値はどう付けるか？→とりあえず1
- 入力データから、特定の要素だけ取り出す
  - 患者ID, 医薬品コード, 評価値
  - 分類番号の前6文字(傷病名の分類), 医薬品コード, 評価値
- Webサーバのアクセスログなどにも適用できる
  - 127.0.0.1 – **aramoto** [26/Sep/2011 11:00:00 +0900] “GET /hoge/hoge/**index.html**” 200 564 – “Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1...)”
  - 127.0.0.1 – **aramoto** [26/Sep/2011 11:00:05 +0900] “GET /hoge/hoge/download?uuid=**1234-5678-9abcd**” 200 24271 – “Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1...)”

- 複数の入力ファイルをJOIN
  - REDUCE-SIDE JOIN, MAP-SIDE JOIN
    - 参考URL: [http://code.google.com/p/try-hadoop-mapreduce-java/source/browse/trunk/try-mapreduce/src/main/java/jp/gr/java\\_conf/n3104/try\\_mapreduce/JoinWithDeptNameUsingReduceSideJoin.java](http://code.google.com/p/try-hadoop-mapreduce-java/source/browse/trunk/try-mapreduce/src/main/java/jp/gr/java_conf/n3104/try_mapreduce/JoinWithDeptNameUsingReduceSideJoin.java)
    - キーがintなので、longに修正
  - MEMORY-BACKED JOIN
    - HDFS上のファイルをメモリに取り込む

```
userfile = `hadoop fs -cat hdfs:///work/output_uid/part-*`  
users = Hash[*  
(userfile.gsub("¥t", "").gsub("¥n", "").split(", "))]
```
- 「ユニークなID」をどう生成するか？
  - mapred.task.id から数字部分だけ抽出し、そこから採番
    - 値: attempt\_local\_0001\_m\_000017\_0
    - id = (ENV["mapred\_task\_id"].scan(/[0-9]/).join(" ").to\_i % 1000000000000) \* 2 \* 1000000



# Rubyによる特定要素抽出(CSV)

```
#!/usr/bin/env ruby
```

```
while line = STDIN.gets
```

```
  line = line.gsub("\n", "")
```

```
  if line.length == 0 then next end
```

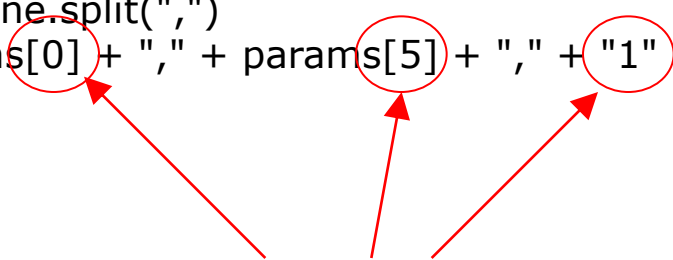
```
  if line[0,1] == "#" then next end
```

```
  params = line.split(",")
```

```
  print params[0] + "," + params[5] + "," + "1" + "\n"
```

```
end
```

必要に応じて変更



- Streamingを使用

- ローカルHadoop

```
hadoop jar $HADOOP_HOME/contrib/streaming/hadoop-0.20.2-streaming.jar ¥  
-mapper "ruby map_uid.rb" ¥  
-reducer "ruby red_uid.rb" ¥  
-input ./input ¥  
-output ./output1
```

- AmazonEMR

```
elastic-mapreduce -j $JOB_ID ¥  
--stream --step-name "map_uid.rb" ¥  
--mapper s3://com.gol.aramoto.aitc/mml/map_uid.rb ¥  
--reducer s3://com.gol.aramoto.aitc/mml/red_uid.rb ¥  
--input hdfs:///input ¥  
--output hdfs:///output1
```



- Mahoutによるレコメンデーション

- － ローカル

- mahout recommenditembased ¥

- input ./output3 ¥

- output ./output4

- － AmazonEMR

- elastic-mapreduce -j \$JOB\_ID ¥

- jar s3://bucket名/jars/mahout-core-0.5-job.jar ¥

- main-class org.apache.mahout.driver.MahoutDriver ¥

- step-name "Recommend ItemBased" ¥

- arg recommenditembased ¥

- arg --input --arg hdfs:///output3 ¥

- arg --output --arg hdfs:///output4

- Mahoutによるレコメンデーションの結果

```

073xxxxx56      [62xxxxx22:1.0,62xxxxx80:1.0,62xxxxx72:1.0,64xxxxx43:1.0,64xxxxx018:1.0,
64xxxxx017:1.0,64xxxxx01:1.0,64xxxxx01:1.0,62xxxxx01:1.0,62xxxxx01:1.0,]
077xxxxx83      [62xxxxx41:1.0,62xxxxx22:1.0,62xxxxx80:1.0,64xxxxx62:1.0,64xxxxx43:1.0,
64xxxxx18:1.0,64xxxxx17:1.0,64xxxxx01:1.0,64xxxxx01:1.0,62xxxxx01:1.0,]
086xxxxx91      [62xxxxx41:1.0,62xxxxx80:1.0,62xxxxx72:1.0,64xxxxx62:1.0,64xxxxx43:1.0,
64xxxxx18:1.0,62xxxxx01:1.0,64xxxxx01:1.0,64xxxxx54:1.0,64xxxxx25:1.0,]
130xxxxx58      [62xxxxx41:1.0,62xxxxx80:1.0,64xxxxx62:1.0,64xxxxx43:1.0,64xxxxx18:1.0,
64xxxxx01:1.0,64xxxxx01:1.0,62xxxxx01:1.0,64xxxxx01:1.0,64xxxxx70:1.0,]
:
    
```

- 各種マスターファイルがWeb上で入手可能

<http://www.iryohoken.go.jp/shinryohoshu/downloadMenu/>

マスター			対象		
マスター名	件数	最終更新日	医科	歯科	調剤
<a href="#">医科診療行為マスター (279KB)</a>	5,705件	平成23年10月14日	○	○	
<a href="#">医薬品マスター (701KB)</a>	18,230件	平成23年 9月30日	○	○	○
<a href="#">特定器材マスター (34.4KB)</a>	971件	平成23年 9月 9日	○	○	○
<a href="#">傷病名マスター (1.00MB)</a>	23,523件	平成23年 9月30日	○	○	
<a href="#">修飾語マスター (43.1KB)</a>	1,979件	平成23年 9月30日	○	○	
<a href="#">コメントマスター (8.46KB)</a>	319件	平成22年 4月 1日	○	○	○
<a href="#">歯科診療行為マスター</a>		平成23年 9月12日		○	
<a href="#">歯式マスター (6.33KB)</a>	915件	平成23年10月 3日		○	
<a href="#">調剤行為マスター (4.68KB)</a>	97件	平成22年 9月22日			○

# やってみて気付いた事

- ローカルのHadoopとAmazonEMRの違い
  - ログがすぐに確認できない
  - IDの素とした `mapred.task.id` のパターンが違う
    - ローカル: `attempt_local_0001_m_000017_0`
    - EMR: `attempt_201109241353_0004_r_000000_0`
  - EMRは「Reduceなし」だとエラーになる
- 変更したrbファイルのS3へのアップロード
  - `s3sync` を使用すると楽だけど
- Mahoutによるレコメンデーション
  - 出力先とtempを毎回削除しないと、エラーになる
    - ローカル: `rm -rf temp`
    - EMR: `elastic-mapreduce -j $JOB_ID --ssh "hadoop fs -rmr hdfs:///user/hadoop/temp"`

- Hadoopは、何をしても分単位の時間がかかる
  - ローカルで効率良く開発
  - ちょっとしたミスがあると、数十分後に後悔
  - 「自動的に実行状況を表示」するツールを作成
  - 「終わったら、EMRをシャットダウン」するツールを作成
  - あいた時間に何をするか考えてから測定
- S3にファイルを置くのに時間がかかる
  - Hadoop distcp, s3sync, Firefoxプラグイン
- 1台での実行に限定すれば
  - ローカルPCで実行した方が早い
  - データ量が爆発した時に必ず困る
  - → 事前にHadoopにしておけば安心

- Amazon EMR+S3の組み合わせは便利
  - S3上に公開データがあれば、とても利便性が高い
  - 複数のEMRを起動し、同時に別条件で測定
  - EMR上のファイルは、EMRを落とすと自動で消える
- 様々なデータをS3に置いて公開して欲しい
  - 防災情報XML、GPV、...

# 今後の展開について

- このモデルに、レコメンデーションは合わない？
  - 実データから抽出したテストデータで実行してみたけど、
    - あまり良い結果は得られなかった
  - お医者さんは、薬の知識がちゃんと頭に入っているはず
    - このシステムを必要とする医者には、診察して欲しくない
  - データ検索や社内文章システムのログ解析では使えそう
- 次は、クラスタリングにトライ
  - 例えば、請求書に書く病名は、
    - 病名によって薬を処方するのではなく、
    - 薬の処方によって、請求書の病名が決まるらしい
    - 過去のレセプト情報を元に、薬の処方から病名の候補を出す
  - さまざまなデータの分類に転用できそう

部会への参加をお待ちしております