协同过滤-矩阵分解

liupeng11

业务

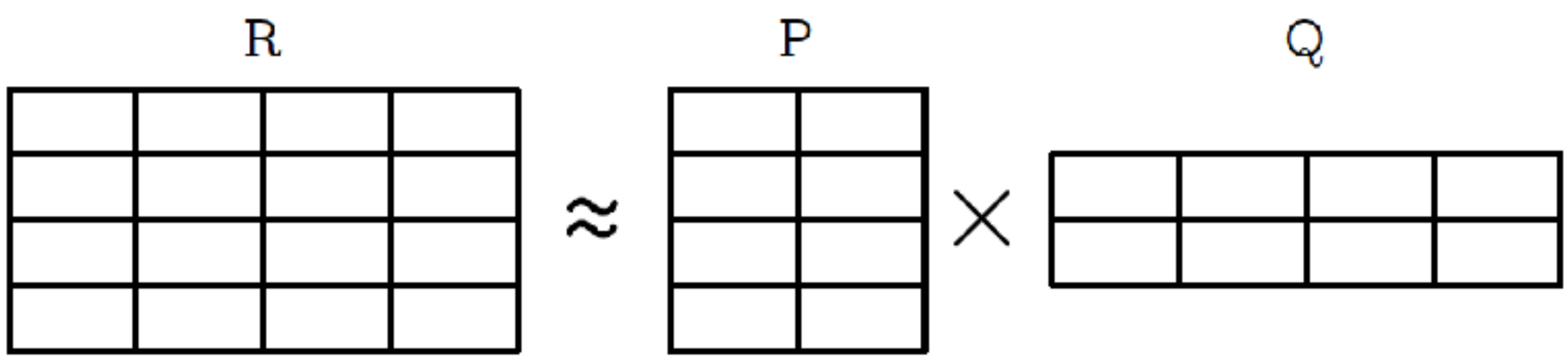
- 画报锁屏场景
  - like
  - unlike(dislike)
  - share
  - buy
  - ~~click~~
- 数据规模
  - (user, image, rating)
  - (3820_8649, 2_7347, 6_5556_0952）
  - 仅 rating, disk parquet 6.5 GB，RDD cache 48+ GB，单机内存不够

常用 CF

- user CF
- item CF
- matrix factorization
  - latent factor
  - 本质降维 user embedding, item embedding

MF

- the completion is driven by a factorization



- associate a latent factor vector with each user and each item
- missing entries are estimated through the dot product

$$r_{ij} \approx p_i q_j$$

MF

- Alternative Least Square (ALS) （using QR， SVD）

  Matrix Algebra

  Theory, Computations, and Applications
  in Statistics

- Stochastic Gradient Descent (SGD)

ALS

- 现成 Spark MLlib ALS
  - 数据结构： CSC-like (compressed sparse column) format matrix
  - 算法：NormalEquation Ax = b solving weighted least squares
  - 优点：肯定收敛且速度快，基本不需要调参
  - 缺点（大数据量）
    - shuffle data size 大，容易 retry -> failed
    - 需要自己实现 continuous training, 花了不少时间 refactor，code

画报 train

- rank 600 （多大可以无误差拟合 train data ?）
- 让最近时间的 rating 有更高权重

  3 epoch for data in (300, 100, 30, 14, 7, 3, 1) 天内

- Spark 600 executors，600 Blocks
- 300天数据量时 1.5 h / epoch，遇到 retry， 此恨绵绵无绝期

画报 recommend

- userFactor dot itemFactor
- userFactor 3 kw, 87 GB
- itemFactor 63 MB，小数据 broadcast

画报写入 redis

- Redis DB, 增大 write timeout，避免写入异常 (AOF 速度跟不上)
- 每个 user 写入 250 个 imageId，占用 12 GB

SGD

- 需要更快的训练速度，更少的计算资源 (一天一算？）
- Distributed Stochastic Gradient Descent

求一阶导

$$L_{\mathrm{NZSL}} = \sum_{(i,j)\in Z} (\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij})^2$$

$$L_{\mathrm{L2}} = L_{\mathrm{NZSL}} + \lambda \left(\|\boldsymbol{W}\|_{\mathrm{F}}^2 + \|\boldsymbol{H}\|_{\mathrm{F}}^2 \right)$$

$$\frac{\partial}{\partial \boldsymbol{W}_{ik}} L_{ij} = -2(\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij})\boldsymbol{H}_{kj} + 2\lambda \frac{\boldsymbol{W}_{ik}}{N_{i*}}$$
$$\frac{\partial}{\partial \boldsymbol{H}_{kj}} L_{ij} = -2(\boldsymbol{V}_{ij} - [\boldsymbol{W}\boldsymbol{H}]_{ij})\boldsymbol{W}_{ik} + 2\lambda \frac{\boldsymbol{H}_{kj}}{N_{*j}}$$

<u>单机计算 demo</u>　　<u>矩阵计算加速</u>

参考论文

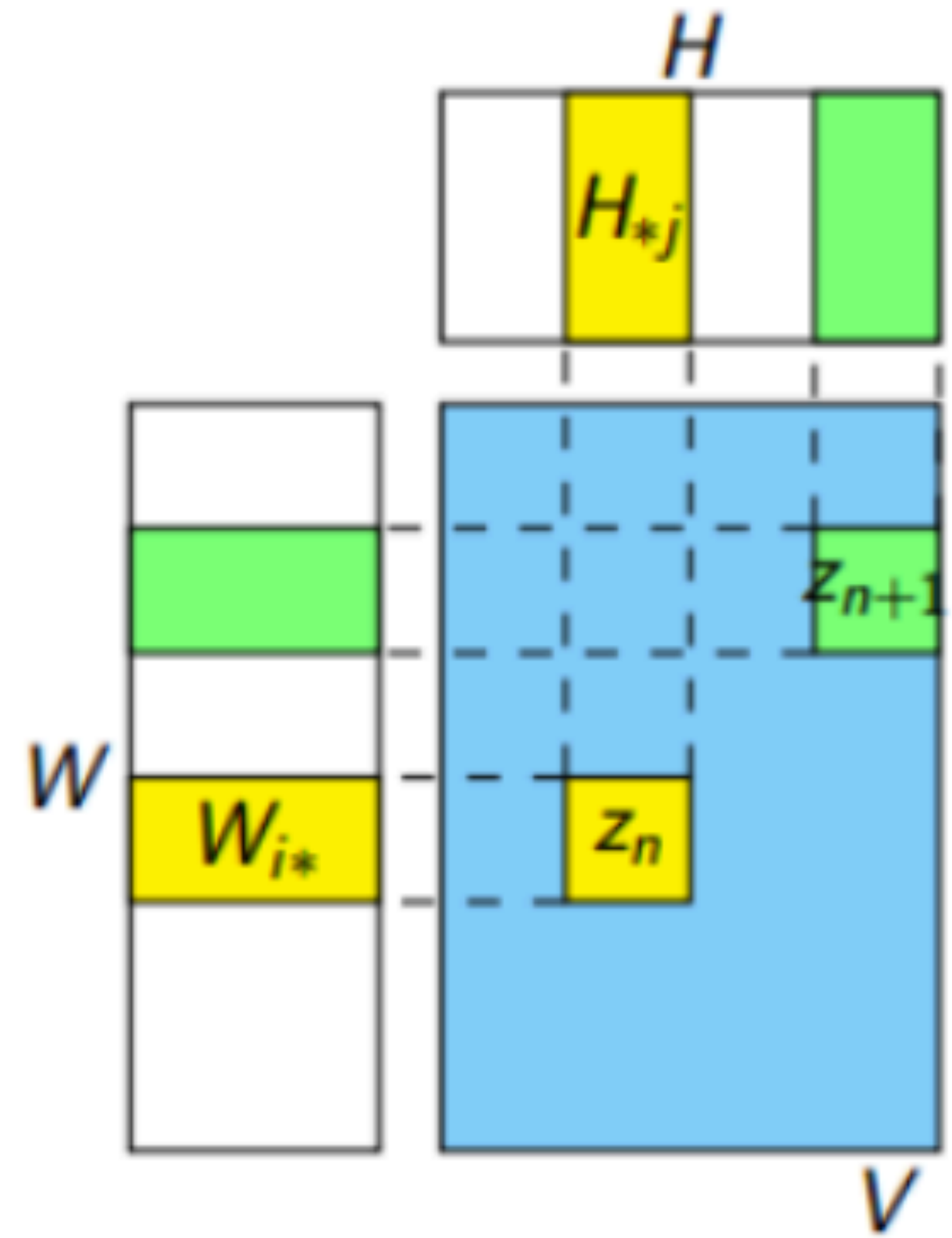
- Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent
- Sparkler: Supporting Large-Scale Matrix Factorization
- 搜不到能用的代码，email 无果，自己实现

Stratum

- 分层训练，如何分层

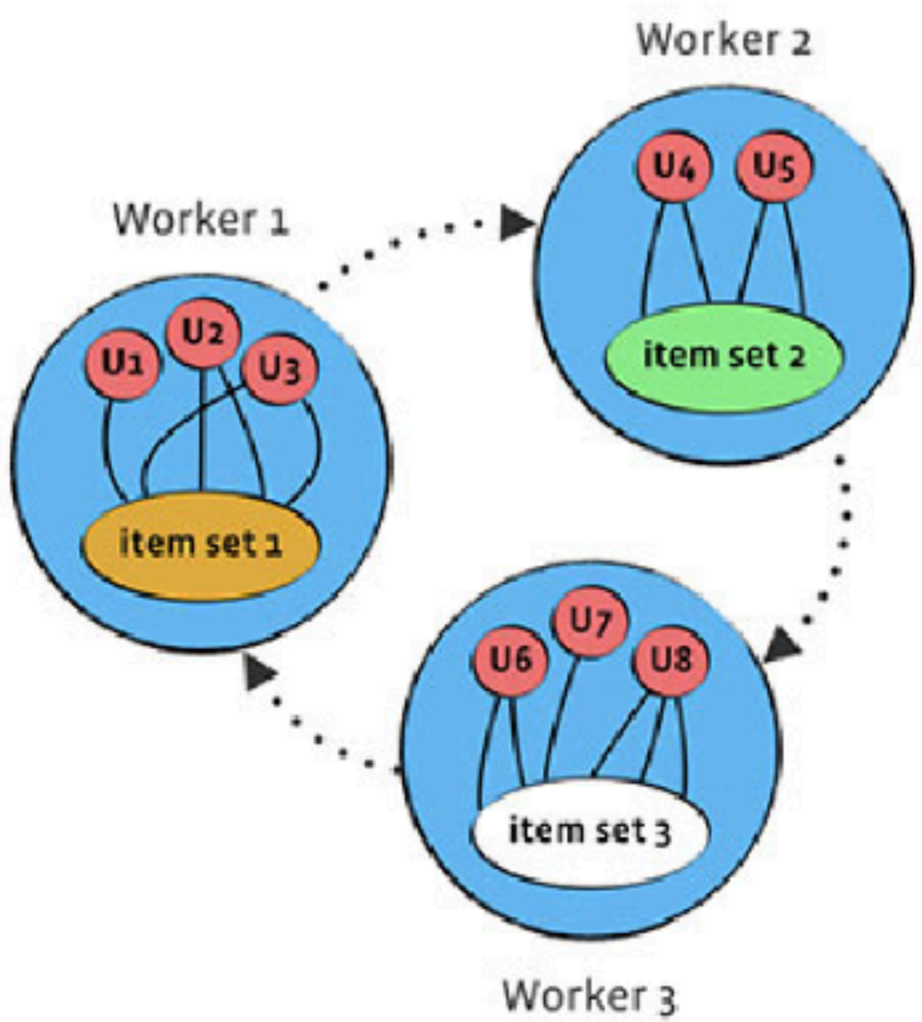
Distributed Stochastic Gradient Descent DSGD

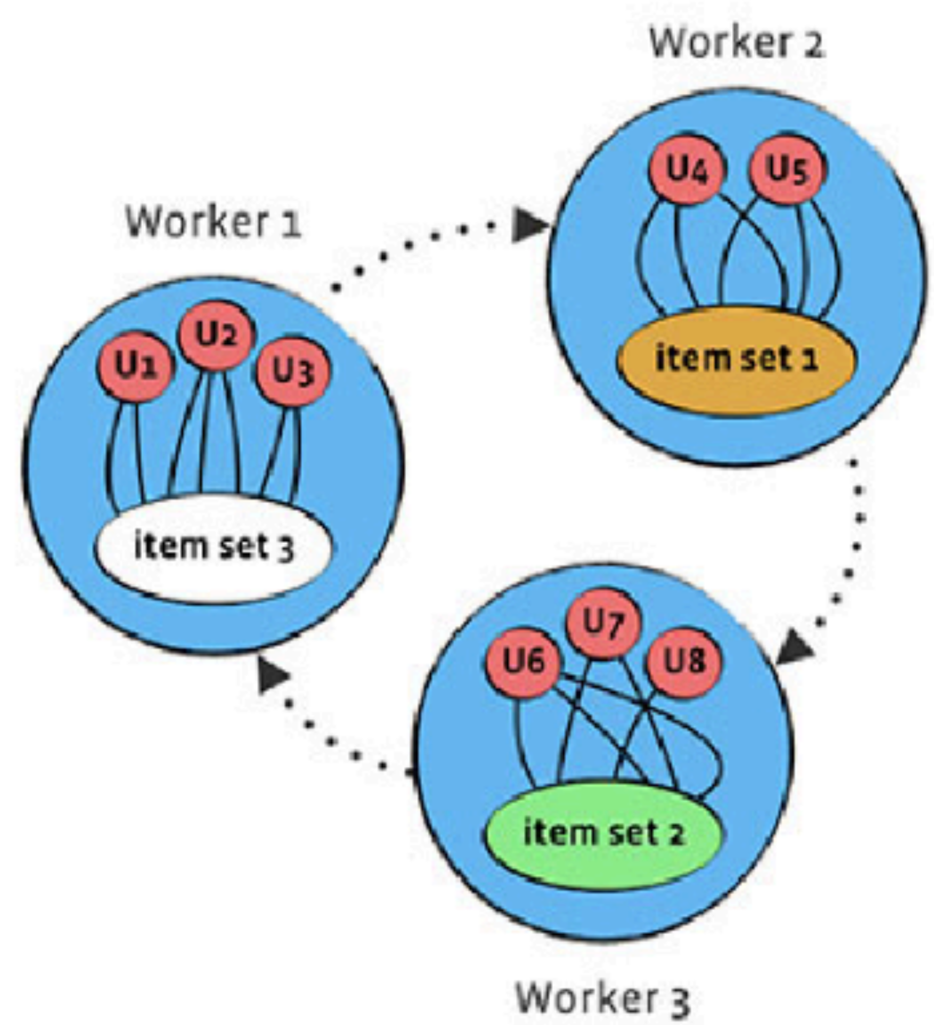
- Divide into interchangeable strata
- d independent map tasks:
  - Each takes a block: Z^b, W^b, H^b
  - Local SGD on each stratum
- Local losses sum
- **Representation** allows parallelism



mapreduce 如何实现

- 参考 recommending-items-to-more-than-a-billion-people





Spark 如何实现

- 怎样做 rotation
- 使用矩阵运算加速 （mutable or immutable？）

code Spark 用好不容易（co-located, co-partitioned）

MovieLens 20M Dataset

```
numRating 20000263, numUser 138493, numItem 26744, numFactor 10, numPartition 3
loss numPart 10 numFactor 10 numPartition 3 in iter 0 time 426 totalTime 426.0 = 1.5281485834675044
loss numPart 10 numFactor 10 numPartition 3 in iter 1 time 543 totalTime 969.0 = 0.7836919719067342
loss numPart 10 numFactor 10 numPartition 3 in iter 2 time 505 totalTime 1474.0 = 0.7093121321246176
loss numPart 10 numFactor 10 numPartition 3 in iter 3 time 585 totalTime 2059.0 = 0.6800817107078139
loss numPart 10 numFactor 10 numPartition 3 in iter 4 time 485 totalTime 2544.0 = 0.6599483589431688



loss numPart 10 numFactor 10 numPartition 3 in iter 5 time 577 totalTime 3121.0 = 0.6455088721581855
loss numPart 10 numFactor 10 numPartition 3 in iter 6 time 574 totalTime 3695.0 = 0.6354341583792952


loss numPart 10 numFactor 10 numPartition 3 in iter 7 time 515 totalTime 4210.0 = 0.6279911761526564



loss numPart 10 numFactor 10 numPartition 3 in iter 8 time 485 totalTime 4695.0 = 0.6218003428483598
loss numPart 10 numFactor 10 numPartition 3 in iter 9 time 534 totalTime 5229.0 = 0.6160719719232891
```

对比测试

- 很不理想
- 训练数据大，存储需要更多 node，但 epoch = subepoch * node_num
- 不能扩展，相比较计算耗时，shuffle 耗时几乎可以忽略

展望

- 能力有限，累觉不爱
- 更快的矩阵运算方法，更好的数据结构
- GPU