Tapestry

人类历史上的第一推荐系统

Zhang Chen

Ping An

2020 年 12 月 30 日

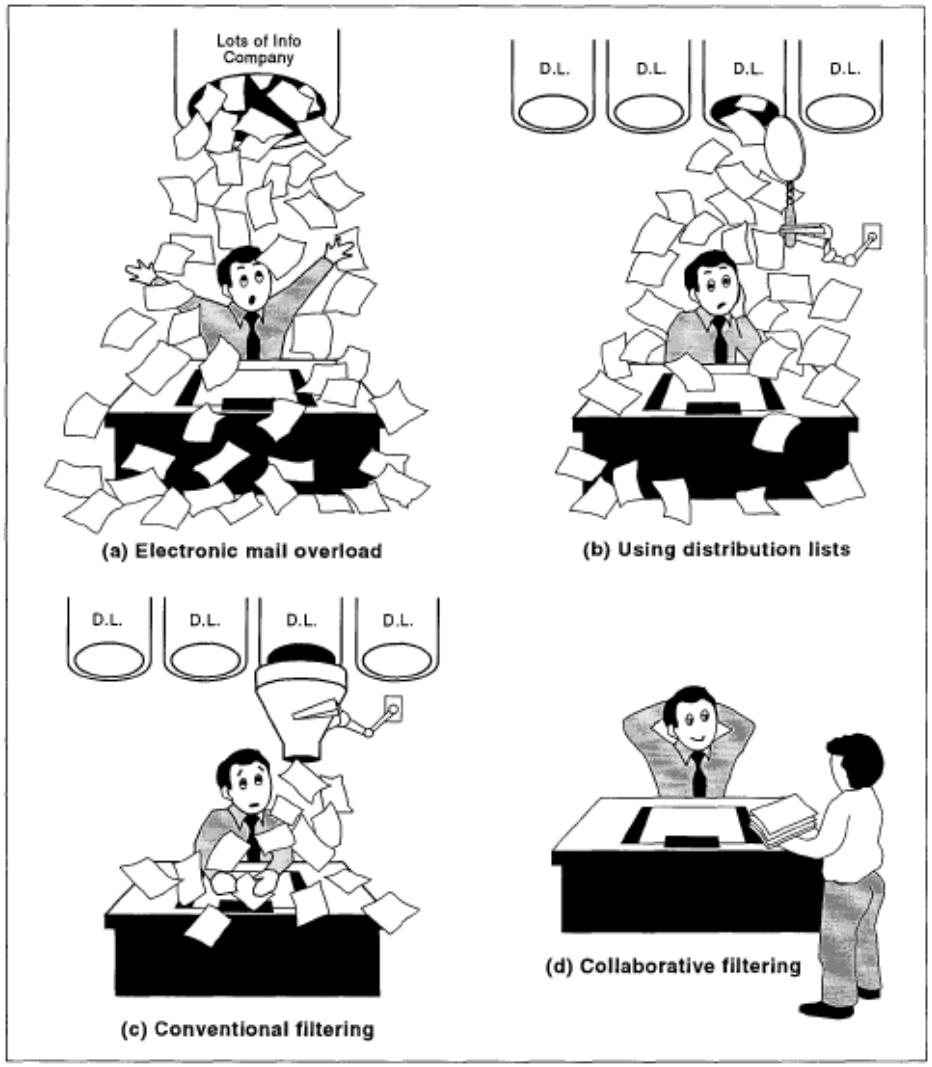
PARC

PARC（Xerox Palo Alto Research Center）

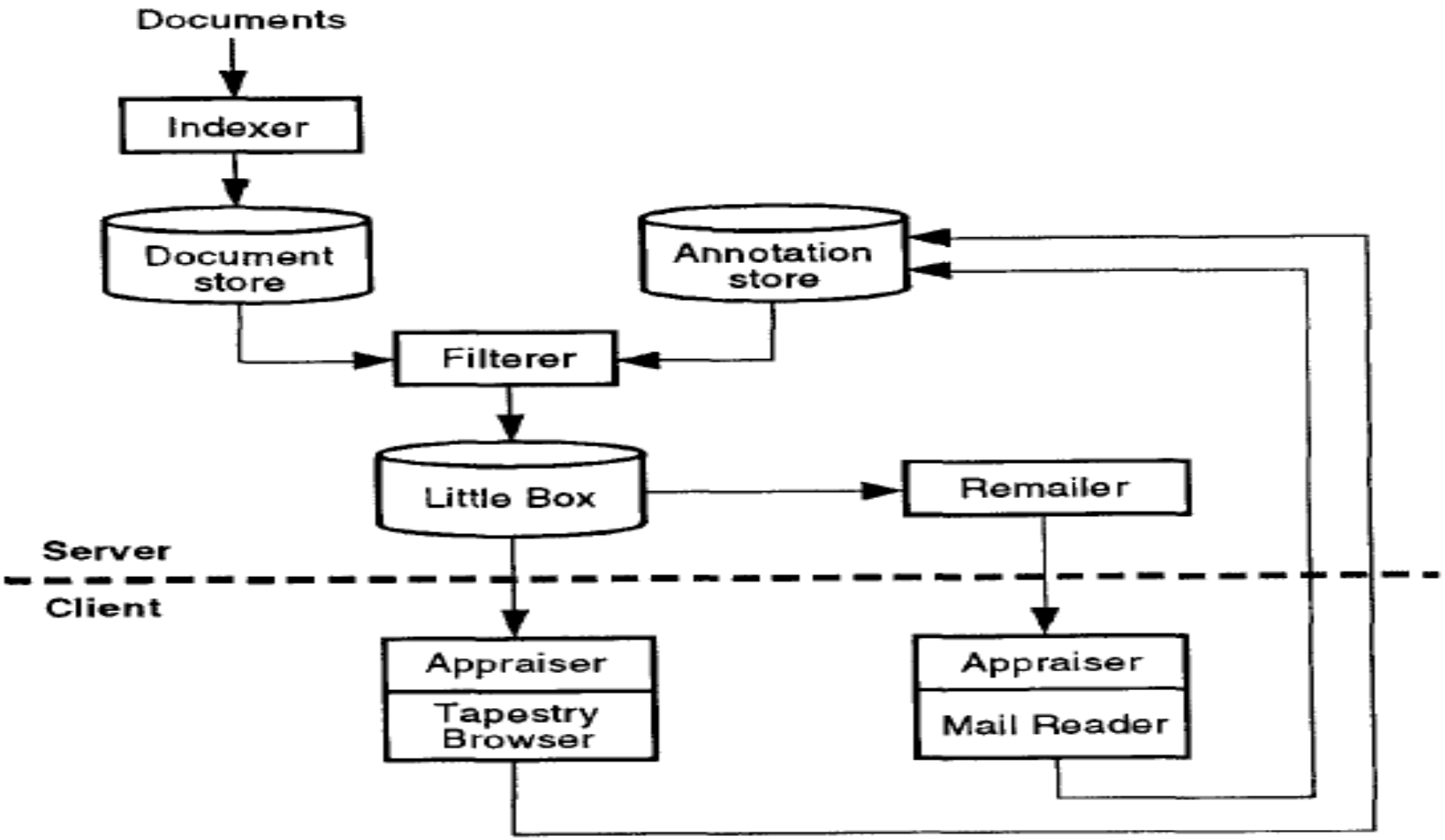
- 1970 年成立
- 1971 年 11 月研制出世界第一台激光打印机
- 1973 年世界第一台具备现代图形用户界面的操作系统
- 1976/1979 年以太网及 CSMA/CD 在这里诞生
- ... 数字视频、文字处理...
- 1992 年 Goldberg、Nichols、Oki、Terry 等人在《Communications of the ACM》发表论文：《Using Collaborative Filtering to Weave an Information Tapestry》

背景

- 上世纪 90 年代，电子邮件广泛普及，信息爆炸，美帝人民水深火热
- 邮件列表-> 过滤器-> 协同过滤
- 人的参与可以使信息过滤更高效



系统架构


Documents
Indexer
Document store
Annotation store
Filterer
Little Box
Remailer
Server
Client
Appraiser
Tapestry Browser
Appraiser
Mail Reader

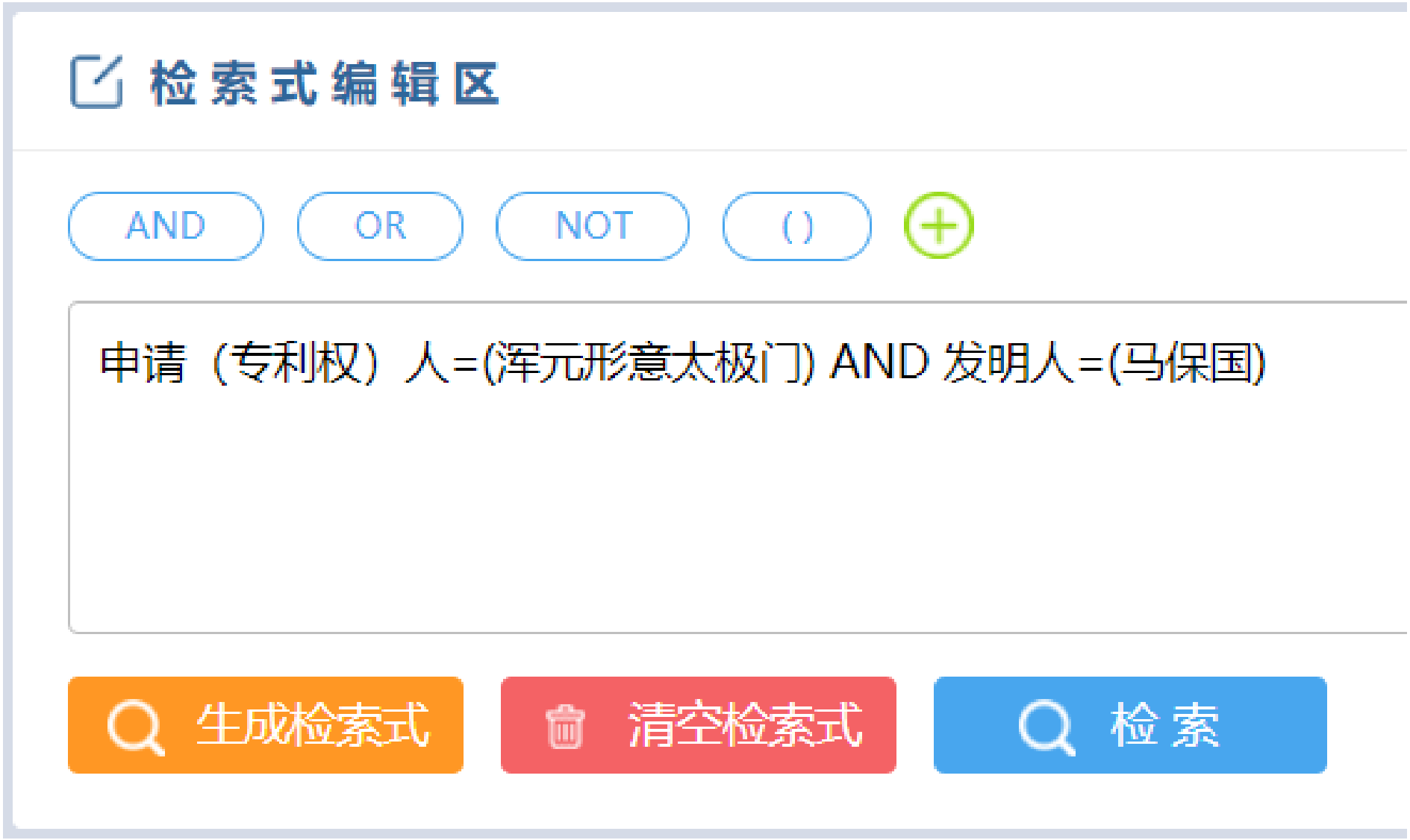
Filter Query

- 如何编写？
- 如何执行？

TQL（Tapestry Query Language）

- Why Not SQL ?
  - 优点：通用，简化设计
  - 缺点：普通用户学习成本高，尤其是面对复杂的 Schema 时
- TQL：简化的领域特性语言
  - 极大简化用户的学习成本
  - 由系统“编译”为 SQL 后执行

降低用户学习成本



降低用户学习成本: 更进一步

耗子尾汁


搜索式
列表式
多图式
申请日降序
过滤
一种松果弹抖闪电鞭的发功方法
【公开】
同族：0
引证：0
被引：0
申请号：CN202010897005.4
申请日：2020.08.31
公开（公告）号：CN111923149A
公开（公告）日：2020.11.13
IPC分类号：B27B13/00；B27B13/04；B27B13/16；B27G3/00；
申请（专利权）人：浑元形意太极门；
发明人：马保国；
代理人：廉海涛;
代理机构：山东甚么事专利代理事务所(特殊普通合伙) 42242;
详览
收藏
+ 分析库
申请人
法律状态
监控

TQL v.s SQL

```
EXISTS(m1:((τ < m.ts AND m.ts ≤ Now()) OR
           (τ < m1.ts AND m1.ts ≤ Now())) AND
           m1.sender = "Joe" AND
           m1.in_reply_to = {m})
```

TQL

SQL

```
SELECT m.id FROM msgs m WHERE
  EXISTS(SELECT * FROM msgs m1 WHERE
    ((@tau < m.ts AND m.ts <= getdate()) OR
    (@tau < m1.ts OR m1.ts <= getdate()) AND
    (m1.sender = "joe") AND
    EXISTS(
      SELECT * FROM reply_to t1, msgs tm1
      WHERE t1.id = m1.id AND t1.reply_ref = 1
      AND
        t1.msg_id = tm1.msg_id AND
        tm1.id = m.id))
```

TQL 基本语法

- 类似一阶谓词
  - 原子表达式 $(=, >, <,$ LIKE , IN)
  - 布尔操作符 (AND, OR, NOT)
  - 限定词 (EXISTS)

TQL 示例

查询主题为马保国的文档

m.subject = '马保国'

一个朋友询问发生什么事了

m.sender = '一个朋友' AND m.subject LIKE '

发给英国大力士和某某小伙子

m.to = {'英国大力士', LIKE '%小伙子%'}

得到马老师回复的文档

EXISTS (m1: m1.send = '马老师' AND m1.in-reply-to = {m})

隔壁老王的查询结果中包含闪电鞭的文档

m IN OldWang.funnyQuery AND m.words = {'闪电鞭'}

TQL 示例（续）

得到马老师点赞的文档

a.type = 'vote' AND a.owner = '马老师' AND a.msg = m

等价于:

EXISTS (a: a.type = 'vote' AND a.owner = '马老师' AND a.msg = m)

优先级为 10 的文档

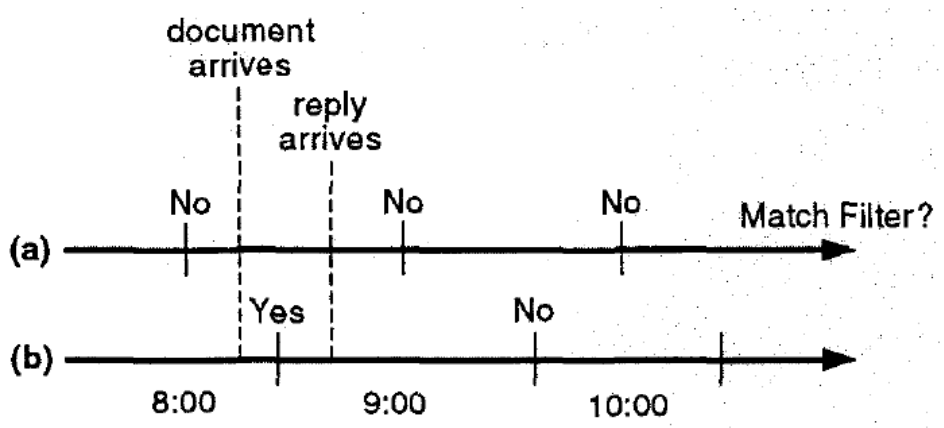
a.type = 'priority' AND a.value = 10 AND a.msg = 'm'

过滤器: 周期执行

- 多查：每次查询的结果都包含了上次查询结果，需要将这部分重复的文档剔除
- 漏查

漏查

- 考虑查询：“所有未被回复的文档“
- 假设某文档 m 在 08:15 到达，08:45 被回复
- 执行周期为 1h
- **薛定谔的查询!**



连续语义

- 查询在时刻 t 执行的结果为 $Q(t)$
- 连续查询语义下，时刻 t 的查询结果为 $\bigcup_{s \leq t} Q(s)$

不适合作为过滤器的查询

- query=”未被回复的文档”
- 在连续语义下相当于“所有未在**第一时间**被回复的文档”——返回所有文档!
- ”**永远**未被回复的文档”如何?——系统需要永远等下去
  - 等待一个较短的时间（如 2 周）就可以覆盖大多数情况
  - 一个邮件要么在 2 周内被回复，要么就永远不会被回复了
- 对过滤器来说更有意义的查询也许是“未在两周内被回复的文档”

临时查询（ad hoc query）

- 由用户输入的“一次性”查询
- 临时查询可以利用存储在客户端的 private fields

连续语义：实现

- 每一个时刻都执行（不现实，期待量子计算机的惊喜）
- 约束条件下更高效的实现方式
  - 只要一个查询的结果集**随时间不减**，则只要**周期性执行**此查询即可满足连续语义
  - ”未被回复的文档”不符合不减的条件，因此粗暴地周期执行就会招来薛定谔
  - ”未在两周内回复的文档”符合条件，可以周期执行
- 通过对查询进行改写实现连续语义

两阶段改写

- **阶段 1**：改为单调形式
- **阶段 2**：改为增量形式

单调形式

- 结果集随时间不减的查询称为单调查询（monotone query）
- 单调查询满足条件：

$$Q^M(t_1) \subseteq Q^M(t_2), t_1 < t_2$$

- 在时刻 t 只需要返回 $Q^M(t) - Q^M(\tau)$ ，其中 τ 为上一次过滤器执行的时间
- 直接计算 $Q^M(t) - Q^M(\tau)$ 是低效的

增量形式

- $Q^I(t,\tau) = Q^M(t) - Q^M(\tau)$
- 不同的执行周期只会影响每个 batch 的数据，但不会影响最终的结果
- 通过索引和查询优化，增量查询的开销与执行间隔内新增数据集而非整个数据集的大小成比例

两阶段改写：示例

Q	m.sender = 'Joe'
Q^M	m.sender = 'Joe'
Q^I	m.sender = 'Joe' AND (m.ts $>$ τ AND m.ts $\leq$ t)

两阶段改写：示例

Q	m.to = 'BugReports' AND m.ts + [2 weeks] < now() AND NOT EXISTS (mreply: mreply.in_reply_to = {m})
Q^M	m.to = 'BugReports' AND m.ts + [2 weeks] < now() AND NOT EXISTS (mreply: mreply.in_reply_to = {m} AND mreply.ts < m.ts + [2 weeks])
Q^I	m.to = 'BugReports' AND m.ts + [2 weeks] > τ AND m.ts + [2 weeks] ≤ t AND NOT EXISTS (mreply: mreply.in_reply_to = {m} AND mreply.ts < m.ts + [2 weeks])

整体流程

Set $\tau = -\infty$
FOREVER DO
  set $t :=$ current time
  Execute query $Q^I(\tau, t)$
  Return result to user
  set $\tau := t$
  Sleep for some period of time
ENDLOOP

Appraiser

- Andrew Message Reader by CMU
  - 支持一种叫做“FLAMES”的语言
  - 将符合条件的邮件放入指定的文件夹
- 邮件阅读器 Walnut
  - 支持自定义规则及得分
  - 命中多个规则得分求和
  - 将邮件归类到不同的文件夹下并按照得分排序

总结

- 主要贡献：提出了**协同过滤**的思想
  - 在过去达成一致的用户很可能在未来也会达成一致
  - eager user v.s causal user
- 古今对比
  - Tapastry: 半自动、透明
  - 现在：全自动、黑盒
- 推荐系统的问题
  - 扎克伯格 (facebook 创始人)：你院前奄奄一息的松鼠可能比挣扎在生死边缘的非洲人民与你的兴趣更加“**相关**”
  - Eli Pariser(Upworthy 致力于用技术建立更好的世界)：两个人同时使用 Google 搜索“埃及”这个关键词，却得到了完全不同的内容，其中一个人得到大量关于游行抗议的结果，而另一个人看到的是一片歌舞升平

参考文献

[1] Using Collaborative Filtering to Weave an Information Tapestry
[2] How recommender systems make their suggestions
[3] Beware Online Filter Bubbles
[4] Continuous Queries over Append-Only Databases
[5] An Overview of the Andrew Message System
[6] Browsing Electronic Mail: Experiences Interfacing a Mail System to a DBMS

Thank You