How to design a web crawler?

* 考察角度 有很多，可以从 系统角度 考虑，也可以从 算法 角度考虑
* 先设计 简单的 单线程，再进化为多线程
* What does Google Crawler do?
  + 收集信息，抓取一个 URL 对应的 网页内容 （字符串）， URL 万亿级别。为什么要 crawl -- 做 网页搜索的 前提 -- 收集到这些信息后 再 进行 page ranking／MapReduce 生成 对应的 搜索关键词 配对
  + 生成一个表格： URL 以及对应的 网页内容， 用来进行搜索
* Scenario：
  + 给定 a list of URL （e.g. wikipedia.org), 里面会有 指向 日本 网站的 URL。这个 graph 是 有向图，万亿级别 (1 后面12个 0， 1 trillion)，进行宽度优先搜索。
  + crawl all the pages in a week
  + QPS：How many web pages? crwal 1 trillion（1000 Billion 1万亿） web pages every week
    1. crawl 1.6Million web pages per second （1 \* 10^12 / 86400 / 7）。1.6 M QPS given
       - 1 trillion web pages
       - crawl all of them every week
    2. 10p (petabyte) web page storage （1petabyte = 10^12 KB）， given
       - average size of a web page: 10k
* 最简单的web crawler：
  + e.g. 如果 最简单，抓取新闻网站的新闻**标题** (tech.163.com/it)
    1. Ruby, Python, NodeJS 抓取网页的content， get 到 content
    2. Regex: 正则表达式。用来进行 字符串匹配。h3 里包着的<a> 标签（链接）anchor tag。获取其内容。返回了新闻标题的数组。发现只有 新闻标题 是 h3 里面 包含的 a 标签
  + e.g. Dropbox 面试题：抓取网页内容 program a web crawler, make it multi-threaded.
    1. single-threaded: 图的宽度优先搜索 - 队列里最开始有A list of URL （会有几个出发点）
    2. while 循环，取出队首的 URL，load web page，写到 磁盘里。 通过正则表达式，提取出每个URL中网页内容的a list of URL，放到 队列里
    3. Given some time， 一定能把所有的网页都覆盖
    4. 整个互联网，有几个URL只有互相指向。考虑孤立节点。伪代码 (见 PPT)

while (queue not empty) {

deque

load, get a list of URL

add 到 队列中

}

* + 1. Producer / Consumer Pattern:
       - 什么时候需要 producer / consumer: pruducer 和consumer的速度不一样。
       - 当前这个 crawler 生产的 速率 高于 消耗的 速率
    2. Snapchat有一道题是写Producer/Consumer 的 code，
       - 一个threadproduce， 一个thread consume，解决好multi-threading的问题
       - reading list 里有具体怎么 写 producer / consumer
    3. 单线程问题：
       - 太慢：t1时刻，发送http request，t2 才返回(中间经历了 DNS， load balancer，等待了几百毫秒），然后再regex取出所有的link -- problem: 在等待http返回的同时，pc是闲着的。 如果是多线程，就可以在几乎同样的时间处理n倍的crawling
* Multi-threaded Web Crawler （on a single machine）
  + Challenge: 多个线程thread同时访问同一个URL队列
  + 多线程 经常会遇到的问题： 资源冲突的 问题：
    - 如果大家都同时读，几乎不会有问题。
    - 如果有的读，有的写，就会有问题， 需要有 (exclusive write)
  + 三种方法来解决多线程对资源**写**操作:
    - sleep: 不着急，发现现在 不 available，过会再来看（问题: 无法在资源free了之后立即过来）
    - condition variable：站在门口等着，资源free了马上就去抢。抢不抢的到再说。
    - semaphore：有一扇门，门上挂了5把钥匙，允许多个线程共同写资源，但是仅允许一定数量的线程。看见门上挂了钥匙，再进去。 semaphore == 1时，效果和 condition variable 就一样了。
    - producer ／ consumer 例子里有具体code怎么实现
  + 是否是线程越多越好？ Not Necessarily，会受到各种各样的限制
    - Context Switching cost (CPU number limitation 单个CPU同一时间只能执行一条指令) 切换时 要保存之前的状态。 每个线程有其程序执行的状态，切换的话需要 保存原来的 状态，并且 L2cache 本来已经加速了， 现在又要重新进行 缓存
    - thread（port） number limitation: HTTP请求其实是一个TCP/IP 的 call，会占用本地的一个port (source port -> dest port). 理论上有 2 byte， 16位， 0 - 65535 个 port （如果不考虑 reserved port）。不可能发起10万个 http call。
    - network bottleneck for single machine：网络带宽会限制。
    - 基于这么多的问题：
      * 我们的解决方案: 分布式 Distributed System
* Distributed Multi-threaded Web Crawler
  + 多个机器 协同工作
  + 需要一个 shared resource。一个machine放所有的URL队列，另外的机器访问这台机器。
    - 问题： 之前的话单台机器的URL队列在内存里，内存大小有限制， 每个crawler 的生产速率大于消费速率，直到crawl了全球的所有url，一台机器不可能有这么大的 内存。
      * 解决方法： 存在磁盘里，放到数据库里。甚至可以考虑用 分布式 数据库。
  + 数据库 （task table）的设计：达到的效果应该和内存中的队列一致。如果考虑 scale，用 NoSQL会比较方便。
    - id, URL, **State**(idle/working，如果开始 抓取了，就标记为 working，让后面的 thread 不要读取当前的 URL), **priority**(0/1/2某些网页优先抓，不是所有网页都是平等的，比如**新闻列表**页面 比新闻内容页面信息量大)， available time (下一次抓取时间 不同的网页更新时间不一样，所以下一次crawl的时间应该也不一样)，还有一个 column: available\_time (下一次抓取的时间，现在暂时不抓取，需要通过这个column 来定义抓取的频率。从宏观来看，要平均每周更新一次URL，从微观看，那些 新闻主页，一定要更频繁的抓取)。
    - WebPageStorage 放到 另外 的 cluster 里面，不和 task table
  + 现在我们已经有了一个 working solution，问题是什么？
    - 单台DB server。假如全球1万亿个URL， 要从中选择 1000个 分配给 machine，这样 select 太慢了。并且还要排序。
  + 怎么解决数据表过大的问题： sharding 分布式数据库，访问数据库前前方会有一个scheduler，用来 分配任务。如果1秒request 1000个任务，10个database，每个db取100个任务，然后再合起来。 这样 SELCT 的 过程就加快了。
  + How to handle content update / content remaining same / content retrieval failure? 如何处理 数据更新，数据抓取失败。使用database table中的available time column。 以2的指数进行调节: 每次crawl之后都更新available time到1周之后，如果一周后发现data update了， 把更新时间缩短一半，不断进行动态调整。如果更新时 发现 HTTP request failure （HTTP response 404/500）或数据没变，则把时间延长一倍。
  + How to handle dead cycle？ (too many web pages in sina.com, the crawler keeps crawling sina.com, and don’t crawl other websites) 图里面的闭环。 强行做一个限制，假设每秒 每个crawler machine 要 1000个任务，任务重最多有 10% 抓取sina.com， 合理分配计算资源。Crawler 的计算资源还要抓取其他的东西
* Scenario: 前面讲过了
* Service:
  + Crawler Service
  + Task Service
  + Storage Service
* Storage: Use Db to store tasks, BigTable to store web pages
* Scale:
  + 单线程太慢single-> multi,
  + 多线程 数量也不能太多， 所以要distributed multi-> distributed,
  + queue 存不下，所以要persisit queue->table, slow select （db sharding）, crawl failure/update handle/ dead cycle （sina.com -> 10%），
  + multi-region 怎么把中国的 数据同步给 美国，可以建立replica在美国（中国也建一个server）

Design Typeahead:

* e.g. Google Suggestion
  + 还没hit enter， 每次输入就可以用预先判断。
  + 跟户prefix -> top n hot key words 根据前缀 来提供 搜索建议
* Scenario： QPS： DAU Daily Average User 500M 5亿 (真实 user 可能在10Blevel)
  + 每人每天搜6次，每次搜索敲4个字符：每个字符都对应一个http request。所以typeahead service每天被使用的次数: 4 \* 6 \* 500M = 12B
  + QPS： 12B / 86400 = 138K （～120K request）
  + Peak QPS = QPS \* 2 = 276K
* Service:
  + 出发点: 网页角度 （从网页角度，我们知道前端发送了一个 HTTP request，需要一个 query service来对数据来源进行查找） 、从数据来源角度（需要data collection service 获取数据来源）
    - Query Service: 前端网页Query这个service: 假设数据已经收集好了，基于收集好的数据，来判断 donald 开头的 搜索词 都有 哪些

keyword / hit\_count

‘amazon’ 20b

‘apple’ 15b

‘adidas’ 7b

‘airbnb’ 3b

如果用SQL语言

SELECT \* FROM hit\_stats

where KEyword Like ‘$(key)%’

ORDER by hit\_count DESC

LIMIT 10

这个query 的问题： databse 表单一定非常巨大， 这个query 效率很低，数据库是用index 来加速的。

第一快是 用 id索引，数字下标特别快

第二快是 name == ‘John Doe’，用 name 找到 index，然后查找

**LIKE 操作更慢**（range query）， WHERE keyword LIKE `abc%` 相当于 where keyword >= abc and keyword <= abd. 相当于 在 binary search tree中找 range 并 order， 这样会更慢

加速： 将

‘amazon’ 20b

‘apple’ 15b

‘adidas’ 7b

‘airbnb’ 3b

变成这样一个 DB：

“a” [“amazon + 20亿次”, “apple + 15亿次”]

“am” [“amazon”]

“ama” [“amazon“， ”amaze“]

“ap” [“ap”]

比如说我们 “a” 开头的 已经有了 10个 元素，如果又来了一个热门词，就替换这10个元素中 的 一个

每个 entry 都放 10个

基于原始表，放到新的表 可能 要花 好几天时间

这样的花，只需要 11 对应，就可以做了更快的 select 操作，就是 where prefix == ‘ama’ 。然后需要前面档一个 cache。

Google 考察的 是 另一种 思路： 加速： 把数据表转换为Trie (内存中）。相比与存储哈希表，可以用 Trie 来存。

* 和 hashMap 的 复杂度一样，都是O(L)的复杂度， 相比于 HashMap， 会比较省空间， 这里我们考虑用 Trie 存

Trie的问题： 单台机器内存可能存不下，单机可能存不下，现在先假设单机可以存下

Query 字符 “a”,问时间复杂度多少？

Trie DFS 遍历 a下面所有节点的 cnt，时间复杂度是多少 -- 假设所有单词 都是英文小写字母 O(26 ^（n- 1）) 。 -- 指数级别，太慢了。。

继续对 Trie进行优化，将Trie转换成 类似与上面直接 记录结果的表 ：

把trie做转换 -- 在**每个节点里** 存 当前字符开始的字符的子树， 每个节点都存一个数组，数组里面只放10个， 表达以当前字符开始的 出现次数最多的节点数目。

存完之后在 trie中也是这样的结构：

“a” [“amazon + 20亿次”, “apple + 15亿次”]

“am” [“amazon”]

“ama” [“amazon“， ”amaze“]

“ap” [“ap”]

空间换时间。多占用空间， 换来了几乎是常数时间的（从指数变成 常数）。

* + - * Trie in memory:
      * Serialized Trie （on disk） then send the response
    - 下面讲：Data Collection Service： 数据的来源: Google 收集来的。Google记录了log data

log data table: 存user，keyword，timestamp 需要用 分布式 存储来存。简单的存储成 HDFS。 这跟word cnt 是一样的，可以用 mapReduce 来统计 每个 单词的 word cnt。

collection service统计当前log data中 每个keyword的Count (可以用map reduce)

每天log data 随时都在更新，collectionService多长时间去更新？

* data Collection Service 和 QueryService 之间如何协同工作：我们希望系统能反映出最近两周的 搜索趋势。
* DataService 可以每两周 对 log data 进行 aggregation，写到巨大一张表上（word cnt）（amazon: 10B, google: 1B etc）
* 然后更新到 Query Service 的 Trie 里
* QueryService 在磁盘中应该有一个Serialized Trie （用以保证断电后 不怕丢） 重启后会deserialize

如何更新 一个 Trie：（假设我们每天做incremental update）

* Live Service 现在有一个trie 用来处理前端请求，我们复制一份，
* 新来的 word cnt 更新这个 offline 的 trie： word cnt 是 incremental 的 （用 200B + 20B= 220B，然后更新 这个过程）
* 更新好offline后，就 switch 开关 切换 这个 新的trie。因为我们不希望在更新的时候 返回的结果不准确。
* 现在有了working solution

优化：前端：How to reduce response time in front-end (browser)

* 如果用户输入d, da, 然后删掉a， 这个时候利用browser caching能够节省一次HTTP request 304。
* 前端：本来说抓 d开头的前10个，但是我们提前获取 da 到 dz分别的前10个， 如果用户 输入了do， 然后再根据 doa，dob...don 搜索1000个

What if a trie cannot fit in memory in one memory?

Distributed. Store trie in multiple machines.

用4台机器共同存超大的trie，把a开头的热门词 就映射到一台机器，

am开头的映射到另一台机器。每个prefix计算hash映射到一台机器上。 映射的算法尽可能取随机分布。尽量让同一个 request的结果在一台机器中找

利用consistent hashing，将前缀prefix对应一台机器。如果用户要查ama，就在根据consistent hashing找到对应的机器上面查

注意并不是 按照 字母 开始 来存

分布式trie的误区：存一台机器，b存一台机器，。。。。这样一定不行。会有非常严重的bias，z开头的一定比c开头的小很多。

DataCollection Service 从 log Data 中去找。500M people，每人每天搜6次，会有 3B个 entry。Log文件的大小： 如果每次都记录每天都会有3B数据写到log中，占用空间越大，越费钱。不是搜索一次，记一次，以概率来进行log，搜一万次才log一次 (随机数0-10000，抽到0才记录)，词频单位变了，但是不影响结果。 但是怎么考虑低词频（1万次以下）词汇：他们就不会被作为热门词。。

log with 1/10000 probability， get a random number

Snapchat的producer consumer问题在 ppt附录里面。