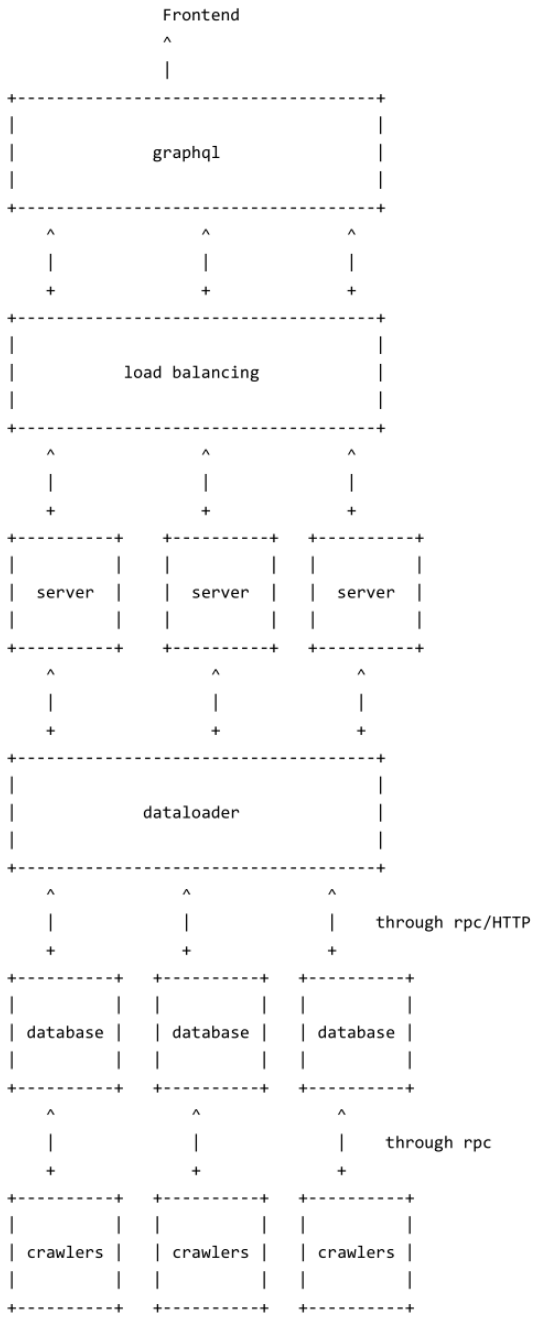
郑和后台计划

zcy

2019 年 10 月 10 日

图 1: 整体架构


Frontend
graphql
load balancing
server
server
server
dataloader
through rpc/HTTP
database
database
database
through rpc
crawlers
crawlers
crawlers

第一部分

数据获取

0.1 使用 scrapy 方案

scrapy 特点

- scrapy 是一个使用 python，未来爬取网站数据，提取数据而编写的应用框架，具有模块化特点。因此只需要几个模块就可以实现爬虫。

scrapy 模块 (主要)

- scrapy engine 负责 Spider、ItemPipeline、Downloader、Scheduler 中间的通讯，信号、数据传递等。
- Scheduler: 它负责接受引擎发送过来的 Request 请求，并按照一定的方式进行整理排列，入队，当引擎需要时，交还给引擎。
- Downloader：负责下载 Scrapy Engine(引擎) 发送的所有 Requests 请求，并将其获取到的 Responses 交还给 Scrapy Engine(引擎)，由引擎交给 Spider 来处理。
- Spider：它负责处理所有 Responses, 从中分析提取数据，获取 Item 字段需要的数据，并将需要跟进的 URL 提交给引擎，再次进入 Scheduler(调度器)。
- Item Pipeline：它负责处理 Spider 中获取到的 Item，并进行进行后期处理（详细分析、过滤、存储等）的地方。

意义

- 开发迅速：一般情况下，只需要对其中的 spider, pipeline 进行编写。
- 适应多网站爬取：爬取数据主要由 spider 进行，如果需要对多个网站进行爬取，可以在同一个项目中快速新建爬虫而不需要修改 Pipeline。因而未必需要通用爬虫。
- 弹性高：易于进行二次调整。

0.2 item

- 此处由杨澍生学长和王清雨大佬达成了对接，目前没有改变。

- 由于要使用二进制传输，所以不需要对 item 进行编写，只要将 data 从 spider 传到 pipeline 并进行打包 item。
- 对这个 item 内容的修改，需要在 pipline 和 sipder 进行微调，不需要对 item.py 进行修改。
- 基于数据库需求的详细要求也由学长完成。

paper *protobuf repeated 类型相当于 std 的 vector，可以用来存放 N 个相同类型的内容。*

```
string title = 1;
string journal = 2;
string journal_name = 3;
repeated string author = 4;
string date = 5;
string pagination = 6;
string issueNumber = 7;
string volumeNumber = 8;
string genre = 9;
```

person

```
string name = 1;
repeated Paper paper = 2;
```

spider 杨澍生学长之前已经完成了对 dblp 的爬虫编写，提供了很好的范本，但是没有实现多网站的爬取。王清雨大佬提供的后台方案中，只需要对爬取的数据打包成 Item 直接存储即可。

未来需要完成的方案

- 根据需求，完成多个网站的 spider 爬虫编写。
- 找出 paper 页面的一般规律，并确保能爬取一个网站的全部内容或者从单一不重复的列表中获取内容。*若能够比较成功地查找 url 的规律，就可以构造 start_urls 来确保内容不被重复爬取。对于能够获得单一不重复列表（例如 DBLP）的网站可以采用此方案。*

- 对于不能找出一般规律的，可以使用半通用爬虫，使用正则表达式或者通过 item 进行判断是否符合需求（比如 item 中发表期刊是否为计算机类型），递归爬取网站全部页面并提取所需要的内容。此处要对需要爬取的网站进行调研。此处使用正则表达式爬取的内容可能会出现数据错误率的问题，因而半通用爬虫可能需要清洗。
- 定期进行重新爬取网站全部内容，并与数据库进行更新。此处考虑可以使用增量爬虫，但是增量爬虫需要两个数据库配合，并且增量爬虫具有内存爆炸的问题，目前只做实验考虑。
- （可能需要）构建 ip 池，对一些可能封杀 Ip 的网站使用。

0.3 pipline

数据库使用 *protobuf*, 因此，只要将爬取的数据加工后直接写入即可。此前，杨澍生学长的爬虫已经完成了对数据地加工。

未来需要完成的部分

- 利用新的接口，将爬虫数据自动写入数据库。

0.4 总结和展望

- 下一步是完成对多个网站数据的爬取。
- 在学长的基础上，这个目标能够比较快的实现。
- 需要加强对爬虫相关知识 (包括爬虫) 的学习，此外，有一些新的内容需要实现需要努力。

第二部分

数据存储

第 1 节　与爬虫对接

需求分析　需要约定接口，对获取到的数据类型及内容进行约束。

Protocal Buffer　是著名科技公司 Google 提出的与语言无关，与平台无关，可扩展的用于序列化结构化数据的机制，类似 `XML` 和 `JSON` 但更简单也更快速。

通过定义`.proto`的接口文件，规定好需要存储的数据的类型、结构、大小等，更方便进行存储。

RPC　同样通过 `Protobuf` 定义 `RPC Service` 需要的参数与格式，供爬虫方调用。

可能存在的问题

- 爬虫获取到不在定义中的数据
- 新的数据与已有数据重复或冲突
- 存储失败后如何重新存储

第 2 节　数据库的选择

需求分析　存储需要做到：

- 体现数据之间的关联
- 可扩展性高
- 可存储的数据量大
- 根据数据间的联系快速搜索

因此我们选用了图数据库 `Dgraph`[1]作为存储数据库。

[1] https://dgraph.io

特点 Dgraph 具有以下特点：

- 分布式——可以部署在多台服务器上，减轻服务器压力
- 快速——能快速对查询进行相应
- 高可用性——自动进行同步复制，重启失败的实例
- 事务支持——保证数据的一致性
- 灵活——随时可调整数据的模式

存在的问题 首先，查询语句不够成熟。

采用了一套自定义的查询语句，不易维护，且语法较为晦涩

其次，该项目比较年轻，不稳定。

主库与从库 由于需要对获取来的数据进行整合、聚类、分析等准备工作，因此设置*主库与从库*，主库中为已经经过处理过的数据，主要用来给用户展示等。而爬虫获取到的数据则直接写入从库，经过处理后再统一与主库进行合并。

采用主库与从库分离的方式，一方面方便对数据的分析与整理；另一方面由主库主要负责读操作、从库主要负责写操作，减轻同时读写对数据库带来的压力。

可能存在的问题

- 如何正确地合并两个库
- 如何保证数据一致性、不丢失
- 如何从从库中获取数据并异步处理

第三部分

数据搜索

第 3 节　两大需求

Aminer 提供快速，精确的搜索服务。以学者搜索为例，系统会按照学者的论文数、被引用数、h-index 等指标排序并且能准确地区分**同名**的不同学者。

而在论文搜索中，Aminer 不仅仅能根据文章名进行搜索，还可以搜索与某一特定主题对应的论文。例如：直接搜索 *Data Mining* 就会出现数据挖掘领域的杰出论文。

因此，郑和平台目前最主要的两个需求如下：

- 根据关键词检索相关学者
- 根据关键词检索相关论文

考虑到用户体验和搜索结果的呈现质量，下述章节将基于这两大需求进行更为细致的分析，同时提供相应的解决方案。

第 4 节　关键词预处理

在查询数据库之前，至少应对用户输入进行如下处理：

- 对用户输入内容的容错（例如拼写错误或输入错误）
- 对同义词、近义词的处理
- 对词型变化及格位变化的处理（词干提取）
- 分词

上述大部分都或多或少地与 NLP 相关。在项目开始的初期（目前），我们考虑使用成熟的开源实现。

4.1　用户输入容错

从后端方向考虑，郑和平台所采用的图数据库内建对 Fuzzy Search 的支持，但需要建立相应的特殊索引。

4.2 同义词与近义词

基于 Word2Vec 的实现方式，进行同义词识别。

同时列出一些较为流行的开源实现以供参考：

- Synonyms[2]，预训练的中文相关词汇实现
- wordvectors[3]，支持多语言的 Word2Vec 实现

4.3 分词

英文语境下的分词依照空格分割之后去除介词和代词。

若来自用户的中文输入为句子（考虑到并非所有用户会用分隔符将关键词分开），首先应进行分词。

开源社区已有极为优秀和高效的中文分词实现 jieba[4]，具有如下优势：

- 基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图 (DAG)
- 采用了动态规划查找最大概率路径, 找出基于词频的最大切分组合
- 对于未登录词,采用了基于汉字成词能力的 HMM 模型,使用了 Viterbi 算法

4.4 词干提取

相比分词，在中文语境下对用户输入进行词干提取的实用意义不是很大（直觉得出，尚无数据佐证）。

而在英文语境下，对关键词进行词干提取是必要的。例如，以关键词 “optimization” 和 “optimized” 分别进行搜索被期待返回接近或者相同的查询结果。事实上，Google 和 Bing 等搜索引擎确实对用户输入进行了词干提取。

采用 Porter 算法 [**?**] 进行词干提取效果较好，论文作者同时提供了免费的改进项目 Snowball[5]。

[2] https://github.com/huyingxi/Synonyms

[3] https://github.com/Kyubyong/wordvectors

[4] https://github.com/fxsjy/jieba

[5] https://snowballstem.org/

第 5 节　权重与优先级

对于每次搜索，用户输入经过预处理后应具有类似如下的结构：

$$i = \{p,\ p[\,],\ s[\,],\ r[\,]\}$$

p 为用户的原始输入字符串，$p[\,]$ 为分词结果，$s[\,]$ 为词干提取结果，$r[\,]$ 为相关词汇。

对 i 中每个元素赋予不同的权重 w_n，依据 $\sum w_n$ 决定搜索结果的相关程度。具体权重的分配方法仍需进一步讨论。

第 6 节　搜索词频统计

在数据库层面进行持久化，对每次搜索过程的分词结果 $p[\,]$ 和词干提取结果 $s[\,]$ 进行索引。

定期分析搜索日志，进行进一步分析（?）

第 7 节　热点数据缓存

为减轻服务器压力，应基于搜索词频统计结果，将热点数据缓存。

详见数据库部分。