23 分布式网络爬虫

更新时间：2019-06-17 19:43:05

> 困难只能吓倒懦夫懒汉，而胜利永远属于敢于等科学高峰的人。
>
> ——茅以升

当面对像豆瓣读书这样数据量巨大的爬取目标并且由于反爬机制的限制，一次性爬网所需要花费的时间是很长的，面对这种大任务我们可以将爬虫化整为零，将爬取工作分散到多台机器上同时进行，本节将介绍如何采用scrapy-redis 将豆瓣爬虫重构为分布式爬虫，加速爬网的效率。

分布式爬网的目的

“大巧不攻，以力破法”

> 制造一个成本极高而无用的天才，不如制造一堆廉价而有用的白痴。

当我们应对一些需要进行长期的增量式爬网的场合，而且目标网站的数量规模极为庞大时，只有一台机器去爬取可能完成每次爬取任务所需的时间就会显得非常的慢长。为了增加爬取的效率我们可以：

- 将一个很长的任务分切成多个时间点来爬取
- 将目标爬取数据进行分区、分块的形式分切爬取任务

以上两点是我们在之前的章节中学到的。

但，如果出现以上两种方法都无法应对的场景时，又或者你想以更短的时间更快的效率完成爬网那么就可以应用爬虫技术中的大招：分布式爬网技术。

分布式爬网是有条件限制的，开发过程并不复杂但是在部署上对于一般的开发人员而言就有点苛刻。要实现分布式爬网的第一个条件是你要拥有**多台具有独立IP且不在同一内部网络下的可运行爬虫程序的机器**，每一台的机器会部署与运行一套分布式爬虫的可执行副本，如果在同一网络下会导致网络出口的拥挤甚至是堵塞。

> 分布式爬网可以说是一种富人级别的技术，“以力破法、以本伤人”，拼的是机器的资源。

因此，这种技术多用于具有一定规模的商业场景。

最简单的分布式爬虫结构

Scrapy并没有提供内置的机制来支持分布式（多服务器）爬取。不过还是有办法进行分布式爬取，这取决于要怎么“分布”了。

如果有很多Spider，那分布负载最简单的办法就是启动多个Scrapyd，并分配到不同机器上。

如果有很多Spider，那分布负载最简单的办法就是启动多个Scrapyd，并分配到不同机器上。

如果想要在多个机器上运行一个单独的Spider，则可以将要爬取的URL进行分块，并发送给Spider。

首先，准备要爬取的URL列表，并分配到不同文件的URL中：

```
http://somedomain.com/urls-to-crawl/spider1/part1.list
http://somedomain.com/urls-to-crawl/spider1/part2.list
http://somedomain.com/urls-to-crawl/spider1/part3.list
```

接着在3个不同的Scrapd服务器中启动Spider。Spider会接收一个（Spider）参数part ， 该参数表示要爬取的分块：

```
$ curl http://scrapy1.mycompany.com:6800/schedule.json -d project=myproject -d spider=spider1 -d part=1
$ curl http://scrapy2.mycompany.com:6800/schedule.json -d project=myproject -d spider=spider1 -d part=2
$ curl http://scrapy3.mycompany.com:6800/schedule.json -d project=myproject -d spider=spider1 -d part=3
```

这种分布式爬虫之间形成的是一个对等网，每个爬虫节点之间没有从属关系，其最大的作用是将爬网任意化，通过控制清单来分配爬网任务。其优点是可以将现有的Scrapy项目直接部署，无须多加改动。但其缺点也是显而易见的——任务列表需要人工分配与更新，可以适用于一些非持久性的轻度增量式爬网场合。

scrapy-redis 的架构介绍

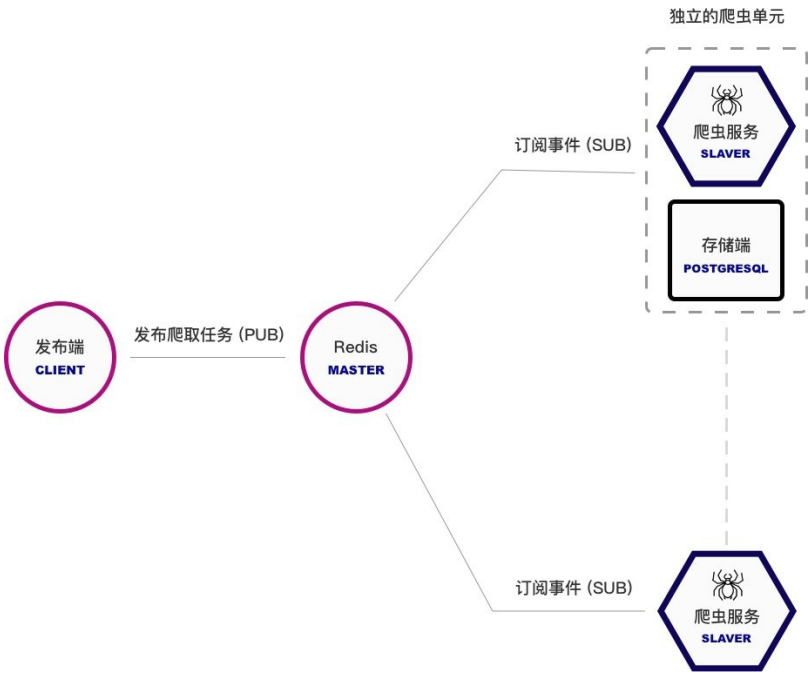
scrapy-redis是一个基于Redis的Scrapy分布式组件。它利用Redis对用于爬取的请求（Requests）进行存储和调度（Schedule），并对爬取产生的项目（Items）存储以供后续处理使用。scrapy-redis重写了Scrapy中比较关键的代码，将Scrapy变成了一个可以在多个主机上同时运行的分布式爬虫。

scrapy-redis分布式原理

假设有4台不同操作系统的计算机：macOS、Ubuntu 14.04、Ubuntu 16.04、CentOS 7.2，任意一台计算机都可以作为Master端或Slaver端。比如：

- Master端（消息服务器） - 使用Ubuntu 16.04，搭建一个Redis数据库，不负责爬取，只负责URL指纹判重、Request的分配，以及数据的存储。
- Slaver端（爬虫程序执行端）：使用macOS、Ubuntu 14.04、CentOS 7.2，负责执行爬虫程序，运行过程中提交新的Request给Master。

- 发布端 — 负责向消息服务器写入爬取任务。这一端可以任何形式的程序，可以是CLI也可以是一个用Flask编写的网站。



其工作过程如下：

1. Master端与Slaver端处于持久运行的服务状态
2. 由发布端向REDIS发起一系列爬取任务（其实就是目标爬取地址）
3. Slaver通过侦测REDIS的事件会从Master端“拿”任务（Request、URL）进行数据抓取，直至REDIS中的已经没有任务为止。

scrapy-redis就是基于以上的原理实现的，每一个爬虫服务就是一个基于scrapy-redis编写的爬虫项目。Scrapy-Redis是在Scrapy进行扩展的的一种用于订阅Redis事件的爬虫与一系列为此服务的扩展包。我们要实现一个分布式爬虫的SLAVER端只要继承scrapy-redis提供的`RedisSpider`来实现蜘蛛、在配置文件中对REDIS服务器进行配置就可以了。

scrapy-redis的工作机理

如果我们自行编写一个分布式爬虫在多台主机上运行，则需要将爬虫的爬取队列进行共享。也就是说，每台主机都需要访问一个共享的队列，然后爬虫从队列中取一个Request进行爬取。当然这些scrapy-redis都已经帮我们做好了，需要做的是如下操作：

1. 初始化爬虫，创建一个Redis的客户端，连接Redis。
2. 查看请求队列是否为空，如果是空则等待，当请求的队列不为空，则从请求队列中拿出一个Request。
3. 获得Request后，经过scrapy-redis的调度器(scheduler)调度后，Scrapy引擎(Engine)会将Request取出，交由下载器中间件(Downloader-Middlerwares)向网站发出请求。
4. 当请求发出并返回响应后，下载器中间件会返回给Scrapy引擎(Engine)，Scrapy引擎(Engine)就会将结果返回至用户写的爬虫(`RedisSpider`)，对结果进行分析处理。产生下一个请求地址又或者直接输出数据项`Item`。
    - 如果得到的是一个Request，则会通过scheduler再次调度，判断Request是否重复，并将Request放入请求队列。
    - 如果已经得到了Item，则Scrapy会将Item交给pipeline处理。

scrapy-redis提供了下面四种组件（components）以实现上述过工作过程：

- Scheduler - 调度器；
- Duplication Filter - 去重过滤器；
- Item Pipeline - 管道；
- Base Spider - 蜘蛛基类。

Scheduler - 调度器

Scrapy改造了Python本来的`collection.deque`（双向队列），形成了自己的`Scrapy queue`（https://github.com/scrapy/queuelib/blob/master/queuelib/queue.py ），但是Scrapy多个Spider不能共享待爬取队列Scrapy queue， 即Scrapy本身不支持爬虫分布式。scrapy-redis的解决是把这个Scrapy queue换成Redis数据库（也是指Redis队列），在同一个redis-server存放要爬取的Request，以便能让多个Spider去同一个数据库中读取数据。

Scrapy中跟“待爬队列”直接相关的就是调度器Scheduler，它负责对新的Request进行入列操作（加入Scrapy queue），取出下一个要爬取的Request（从Scrapy queue中取出）等操作。它把待爬队列按照优先级建立了一个字典结构，比如：

```
{
    优先级0 : 队列0
    优先级1 : 队列1
    优先级2 : 队列2
}
```

然后根据Request中的优先级来决定该入哪个队列，出列时则按优先级较小的优先出列。为了管理这个比较高级的队列字典，Scheduler需要提供一系列的方法。但是原来的Scheduler已经无法使用，所以使用Scrapy-redis的scheduler组件。

Duplication Filter - 去重过滤器

在Scrapy中用集合实现Request去重功能，Scrapy中把已经发送的Request指纹放入一个集合中，把下一个Request的指纹拿到集合中比对，如果该指纹存在于集合中，说明这个Request发送过了，如果没有则继续操作。

在scrapy-redis中去重是由Duplication Filter组件实现的，它的实现原理与我们之前介绍过的`RedisDupFilter`完全相同，相比之下我们更进一步在此基础上还增加了一个基于REDIS的布隆过滤器，所以我们完全可以跳过这个内置的过滤器对象，用我们开发的功能更强大更实用的布隆过程器。

数据项管道(Item Pipeline)

当引擎将爬取到的Item发给数据项管道(Item Pipeline)处理时，scrapy-redis的内置数据项管道(Item Pipeline)会将爬取到的数据项(Item)存入Redis的Items queue。

其实，这个做法可圈可点，在实战过程中我觉得这甚是一种鸡肋般的存在，原因非常简单，Redis只能支持20G的内存数据库，要实施分布式爬虫网络的项目数据项目远远会大于20G！这种结构只会让整个分布式爬虫网络因为数据量过大而彻底崩溃！所以并不建议使用，要进行分布式爬取就应该采用分布式数据存储结构来处理数据端，或者将数据直接写入数仓等的方法。

scrapy-redis 的蜘蛛

scrapy-redis 不再使用scrapy原有的`Spider`类，要实现分布式爬虫就必须从scrapy-redis提供的`RedisSpider`或`RedisCrawlSpider`中继承。

scrapy的原生蜘蛛在爬取任务完成后就会直接让整个scrapy进程退出，而 scrapy-redis的蜘蛛却完全不同，它是一个永久运行的蜘蛛或者更准确地说scrapy-redis让scrapy爬取变成了一个服务。即使爬取完成了，它就只会继续等待Redis中的下一个爬取任务而不会消亡。

基于 scrapy-redis 改写豆瓣读书爬虫

在安装scrapy-redis之前需要先准备个redis服务器或者虚拟机，以提供Master任务队列服务。然后键入以下的指令安装redis工具包与scrapy-redis。

```
$ pip install redis
$ pip install scrapy-redis
```

配置 scrapy-redis

首先，我们来起动一个redis的docker:

```
$ mkdir data
$ docker run -p 6379:6379 -v $PWD/data:/data -d redis
```

然后，在`settings.py`中添加Redis的配置，使用`REDIS_URL`声明redis服务的访问信息：

```
# Redis设置
REDIS_URL = 'redis://127.0.0.1:6379/0'
```

或者，采用另一种配置方式

```
REDIS_HOST = "127.0.0.1"  # REDIS主机
REDIS_PORT = 6379               # REDIS服务端口
# REDIS_PASSWD = "***"            # 访问密码
REDIS_DB = 0                    # 数据库索引号
```

> 注：`REDIS_URL`的优先级最高，大于`REDIS_*`，两种配置方式只要配置一种就好了。

接下来配置去重、调度器与数据管道：

```
#使用scrapy-redis中的去重组件
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
# 使用scrapy-redis中的调度器
SCHEDULER = "scrapy_redis.scheduler.Scheduler"
# 允许暂停后,能保存进度
SCHEDULER_PERSIST = True

# 指定排序爬取地址时使用的队列
# 默认的，按优先级排序（Scrapy默认），由sorted set实现的一种非FIFO、LIFO方式
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.SpiderPriorityQueue'
# 可选的，按先进先出排序（FIFO）
# SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.SpiderQueue'
# 可选的，按后进先出排序（LIFO）
# SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.SpiderStack'

ITEM_PIPELINES = {
   'scrapy_redis.pipelines.RedisPipeline': 400
}
```

这里最重要的一点是要配置scrapy-redis的`Scheduler`，这可以说是scrapy-redis的动作核心。在上述配置中出现了一个去重过滤器，这是scrapy-redis原生搭载的基于Redis的去重过滤器。既然是使用分布式爬虫，那么目标数据一定极为庞大，所以推荐使用在“高效的Redis布隆过滤器”一节中提到的布隆过滤器。最后启用`RedisPipeline`，将采集后的数据进行存储等后处理。

改写分布式蜘蛛

scrapy_redis的唯一缺陷就是不能通过中间件技术接入到爬虫系统，所有基于scrapy-redis架构下的蜘蛛都必须继承自`scrapy_redis.spiders.RedisSpider`或`scrapy_redis.spiders.RedisCrawlSpider`类。

另外，在蜘蛛中我们将初始URL存放在`start_urls`类成员中。而在`RedisCrawlSpider`类中，我们需要初始化的成员是`redis_key`，这是Redis数据库的一个队列的名。爬虫开始运行的时候，读入`settings.py`中的Redis配置来访问远程的Redis数据库。如果根据`redis_key`从数据库中获取初始的URL来爬取爬去，Redis数据库所在的主机就是Master，所有从机（Slaver）都配置好主机的Redis信息，然后运行爬虫，就能不断地从Redis数据库中获取待爬取的URL。

```
# -*- coding: utf-8 -*-
from scrapy.linkextractors import LinkExtractor
from scrapy.loader import ItemLoader
from ..items import BookItem
from scrapy_redis.spiders import RedisCrawlSpider
from scrapy.spiders import Rule

class BookSpider(RedisCrawlSpider):
    name = "doubanbook"
    rules = (Rule(LinkExtractor(allow=('\/tag\/(.*?)')), follow=True),
             Rule(LinkExtractor(allow=('\/tag\/(.*?)\?start\='),
                                tags=('link'), attrs=('href')), follow=True),
             Rule(LinkExtractor(allow=('\/subject\/.*'), ), follow=False, callback='parse_item'))
    redis_key = 'spider:start_urls'

    def parse_item(self, response):
        # 与原来一至，此处略去
```

> 如果你希望蜘蛛变得更具通用可以将`redis_key`的值通过配置文件来指定，如下所示：
>
> ```
> REDIS_START_URLS_KEY="spider:start_urls"
> ```

分布式蜘蛛的改写仅仅需要将来原继承的基类更改为`RedisCrawlSpider`，同时将`strat_urls`去除掉即可。

此时就可以通过`scrapy crawl doubanbook`来起动蜘蛛，然后会看到爬虫进程会停着不动，这是正常的因为爬虫现在正处于待命状态，等待从REDIS中发来的爬取任务。

编写发布端

那谁来产生`start_urls`的爬网地址呢？存储进Redis`spider:start_urls`键内的值是由外部程序产生的，是不是很奇怪？确实习惯于单机开发的模式之后一下子很难将思路切换过来，由于蜘蛛代码是被分布在各个节点上的，而且是处于持久运行的状态，只有侦测到Redis中的`spider:start_urls`有新的URL值的时候才会运行起来，如果在蜘蛛中写入产生URL的逻辑的，那么每个副本节点在运行的时候都会向Redis写入相同的URL这样就乱套了。所以只能将写入`start_urls`的过程分离到爬虫系统之外，或者是某个脚本代码，又或者可以是某个Web进程。

在本例中我们就将URL的产生保存在一个`pub.py`的python脚本中，在任何能访问到Redis的机器上运行该文件就可以启动整个分布式爬虫网络了。

```
# coding:utf-8
from redis import Redis

redis = Redis(host='127.0.0.1', port=6379, db=0)

# 写入Redis启动分布式爬虫网络
redis.lpush('spider:start_urls', 'https://book.douban.com/tag/')
```

接下来你可以打开一新的终端窗口运行:

```
$ python pub.py
```

你就会看见在待机的爬虫马上就工作起来了!

小结

分布式爬虫的开发就这么简单？对！将前文内容与本节示例归纳一下，可以总结出编写分布式爬虫的几个要点：

1. 部署Redis服务
2. 定义Item
3. 继承`RedisSpider`编写Spider
    - 定义`redis_key`
4. 在`settings.py`加入scrapy_redis要求的配置项
5. 将爬虫部署在多台节点上（对于持续迭代的项目建议使用scrapyd部署）
6. 编写生成起始爬取URL的启动脚本

另外，这个示例里我留下了一些值得我们深思的点，那就作为一个课后的习题让你去发挥吧：

- `pub.py`只向REDIS发起一个任务(起始URL)，当分布式爬虫部署到网络上之后就只能有一只蜘蛛会运行起来，这明显就失去了分布式的意义，那如何能让这种任务分配得当呢？
- 配置中将收集到的`Item`全部通过`RedisPipeline`写进了REDIS内，既然前文提及不要这样用，那么请思考一下如何将这个存储端换成你所熟悉的一种方式：MongoDB或者PostgreSQL？
- 将去重过滤器换成之前章节中介绍的高效的REDIS布隆过滤器对比一下爬取效率？

> 分布式爬虫的源代码我发布在豆瓣爬虫项目的一个scrapy-redis分支上，要切换到该分支，请拉取主分支并进入项目目录后键入以下的指令:
>
> ```
> $ git checkout scrapy-redis
> ```