# 基于 Scrapy 技术的分布式爬虫的设计与优化

刘泽华* 赵文琦 张 楠
LIU Ze-hua ZHAO Wen-qi ZHANG Nan

**摘 要** 随着全球信息技术的发展，互联网中的信息量呈爆炸式增长，人们对信息的需求量也与日俱增，而传统的单机平台的爬虫技术已经无法实现现如今互联网中大量数据的获取。本文旨在设计一个基于 Redis 的主从模式分布式爬虫系统来突破传统单机爬虫的限制。 本文中的爬虫系统基于 python 语言的 Scrapy 框架具体实现。此外，为了实现分布式，该系统还使用了 Redis 数据库进行 url 存储与调度分配，最终使用 MongoDb 数据库储存解析后的网页信息。本文也对该系统进行了一定优化，首先，本文采用半分布式拓扑结构优化了整体主从架构；同时本文也采用二级哈希映射算法优化 URL 的分配，解决了系统内节点动态加入或退出对系统的影响；初次之外，本文还采用去重与增量爬取优化了服务器的资源效率并使用代理 ip 的方式来应对部分网站的防爬虫屏蔽现象。

**关键词** Scrapy；分布式爬虫；哈希映射算法；设计与优化；防爬虫屏蔽

doi：10.3969/j.issn.1672-9528.2018.h2.030

## 0 引言

作为一种大量获取网络信息的手段，网络爬虫被人们所熟知。早期的网络爬虫技术一般为单机网络爬虫。单机网络爬虫的原理如下。首先，爬虫系统设置待爬队列，在队列中取得 URL，然后获取该 URL 对应的页面，最后从该页面中取得新的所有的超链接放入待爬队列中，不断循环这些步骤直到将待爬队列中的全部 URL 爬空。然而，互联网近年来的爆炸式发展，导致互联网中的数据量迅速增加，给传统的单机爬虫技术带来了许多新的难题和挑战。即便是成本极高的大型服务器有时也无法处理如此大量的数据。因此，本文采用基于 Scrapy 技术的分布式爬虫设计，设置多个节点对海量数据进行并行获取与处理。该分布式爬虫设计旨在设计一个简单稳定的，中小规模的，主从架构模式的分布式爬虫系统，并提供基本思路与针对其中一些功能的优化方案。

## 1 相关背景介绍

### 1.1 爬虫的基本工作原理

爬虫在工作时，可以将我们要进行爬取得所有网页链接视作树状结构，从一个起始 URL 节点开始，顺着网页中的超链接，不断进行爬取，直至进行至叶子节点。对于爬取超链接的方式，主要有以下两种算法[1]：

1) 深度优先算法：选取最初的一个网页，在该网页中选取一个跳转至下一网页的链接，在该链接跳转至的下一网页中重复该步骤，直至遍历到叶子节点，即网页中不再存在跳转至下一网页的超链接，此时从上一节点选取一个新的链接重复上述遍历过程，不断重复以上步骤，直到将该路线中的全部超链接遍历完成。在此之后，我们将选取另一个初始网页，继续重复以上遍历过程。该算法的优点是较为容易设计网络爬虫。

2) 广度优先算法：该算法是爬虫先获取到初始网页中的所有链接对应的网页，再从中选择一个链接，继续获取该链接对应网页中的所有链接对应的全部网页，不断重复上述过程。该方法是理论上最佳的实现网络爬虫的方法。由于有些网络结构复杂，进行深度优先算法可能导致爬虫系统不断跳转至更深一层的超链接，导致一个分支变得无限长。而广度优先算法可以有效避免这种情况。广度优先算法使爬虫系统并行爬取相同深度的网页，极大提高了信息获取的效率。

我们的基于 Scrapy 技术的分布式爬虫系统设计就采用了广度优先算法进行超链接爬取。

### 1.2 主从架构模式

各大公司使用的爬虫技术系统架构大致可分为三大类：主从模式，自治模式和混合模式[2]。本文将重点介绍其中的主从模式。

* 北京邮电大学 北京 100876

[基金项目]“北京邮电大学大学生研究创新基金”资助（Research Innovation Fund for College Students of Beijing University of Posts and Telecommunications）

主从架构模式指选定一台主机作为Master节点，负责对所有正在运行的Slave节点进行管理调度。每个Slave中的节点只需要获取任务，并且把新生成的任务交回给控制节点，在不与其他Slave节点进行通信的情况下对任务并行处理，这样每个Slave节点都将易于管理。而控制节点则需要与所有Slave节点进行通信以实现管理调度，一般控制节点中都有一个地址列表来记录所有爬虫信息。当系统内爬虫数量变化时，控制节点将更新地址列表信息，该过程对系统中的各个爬虫也是透明的。这种模式具有结构简单，便于增加Slave节点，任务分配效率高等特点。然而，由于架构过于依赖Master节点，该架构也有可靠性低（整个系统的调度与任务分配过度依赖Master节点，如果Master节点出现问题，相应的整个系统都将受到影响），效率低（随着越来越多Slave节点加入系统，对于任务分配的负载将越来越大，Master节点的性能若不能满足任务分配的负载，整个系统的效率都将受到限制）等缺点。

本文所介绍的基于Scrapy技术[4]的分布式爬虫正是一种主从结构的基于广度优先遍历算法设计的爬虫。

## 2 分布式模块系统结构设计

我们可以粗略的将分布式网络爬虫分为两个模块，一个是分布式模块，一个是网络爬虫模块[3]。该部分完成了分布式模块的设计，通过系统架构图详细介绍了本文中基于Scrapy技术的分布式爬虫的系统架构，分析了该系统结构的优缺点。

### 2.1 系统架构

本文所介绍的基于Scrapy技术的分布式爬虫是一种主从结构的基于广度优先遍历算法设计的爬虫，系统分为Client，Master，Slave三个主要部分。其中Client端负责将待爬行任务提供给Master并从Master获取任务结果；Master端通过和Redis数据库交互来寻找待爬网页url，并将其存储至Redis数据库中，同时Master节点按照Slave节点性能将已存储至Redis数据库中的任务分发给各个Slave服务器进行执行，并且实时对Slave进行管理和监控调度；Slave端部署Scrapy爬虫提取网页完成由Master端下发的爬虫任务，并解析提取数据，将其解析的数据储存在MonogDb数据库中。Scrapy爬虫抓取的所有url将根据其key值被分为两类，一类是key值为detail_request的含有我们所需要信息的网页的url，还有一类是key值为next_link 的指向下一网页的跳转链接url。key值为detail_request的url将被储存在Redis数据库中，它们被分为Slave节点待爬取和已爬取的队列，在Slave节点待爬取队列中的url将根据Slave节点性能以及二重哈希映射算法被分配给各个Slave节点进行并行爬取。key值为next_link的url是Redis数据库为Master节点储存的下一轮遍历的网页的链接，这些url将随时传递给Master节点帮助其进行新一轮的遍历。将key值为next_link的url存储进Redis数据库有效的节省了Master节点的资源消耗。爬虫的主要架构如图1所示。

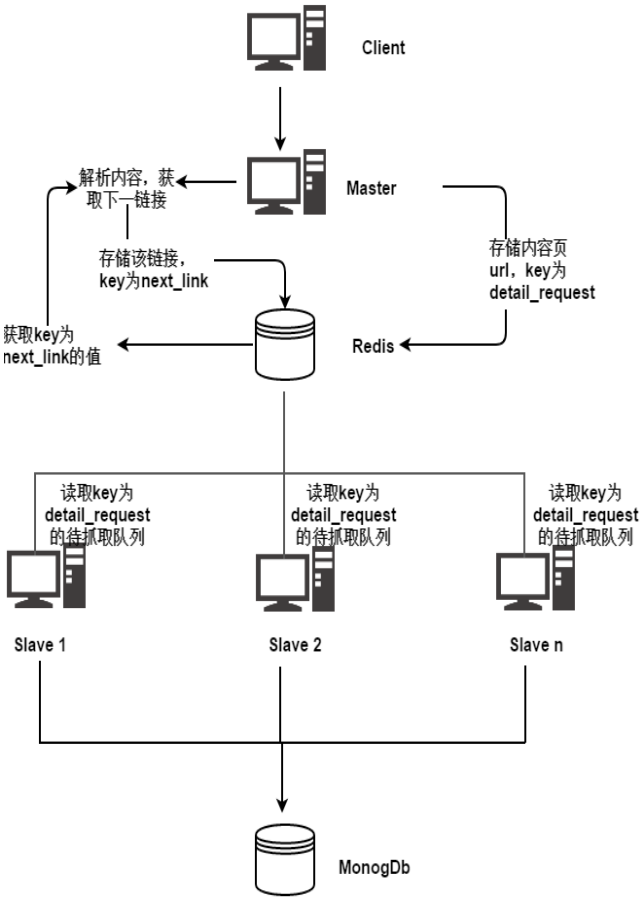


图1. 系统结构图

### 2.2 系统架构优缺点

采用该系统有显著的优势。系统使用了Redis数据库来储存所有url队列，充分利用了Redis数据库存取速度快，易于拓展以及操作简单的特点[5]。系统中，所有Master和Slave节点均可与Redis数据库进行直接交互，大量减少了Master节点本来应该为维护URL队列与维持与Slave节点进行通信付出的资源量。Master节点现在只需将获取到的url队列储存到Redis数据库中，然后按照Slave节点性能分配任务给Slave节点，Slave节点接到任务后，启动本地爬虫，访问Redis数据库获取被分配的url队列然后开始工作。

然而，系统设计仍存在不足。第一，由于系统设计之初采用了主从架构，所以该系统也存在着主从架构的缺陷，即可靠性低（整个系统的调度与任务分配过度依赖Master节点，如果Master节点出现问题，相应的整个系统都将受到影响），效率低（随着越来越多Slave节点加入系统，对于任务分配的负载将越来越大，Master节点的性能若不能满足任务分

配的负载，整个系统的效率都将受到限制）[3] 等缺点。第二，由于该系统处理的信息量很大，我们必须保持整个系统长期稳定运行。在此期间，系统内的 Slave 节点数很可能会发生增减变化（有新加入的 Slave 节点或者现有 Slave 节点发生故障）。然而，普通的任务分配策略可能会导致当某个 Slave 节点故障时重新分配所有任务，或者当新加入 Slave 节点时将已分配任务重新分配给新节点。为了避免这些情况的发生，分布式系统的任务分配策略的算法应该满足动态可配置性 [6]。单纯地根据 Slave 节点性能分配任务的普通任务分配策略将无法灵活应对新节点加入或节点退出系统的情况。

## 3 分布式模块系统的优化

### 3.1 主从模式的优化

针对主从模式的优化，我们可以用半分布式拓扑结构优化简单主从模式。这种结构是中心化结构和全分布式非结构化拓扑结构的结合体。在这种系统结构中，我们可以选择多个性能较高的节点作为 Master 节点，以每个 Master 节点为中心按照图 1 建立分布式爬虫系统。各个 Master 节点之间采用全分布式非结构化拓扑结构连接。这样既保有了系统原有的优点，提升了系统性能和可扩展性，还在一定程度上解决了可靠性低和效率低的问题。其简单结构如图 2 所示。

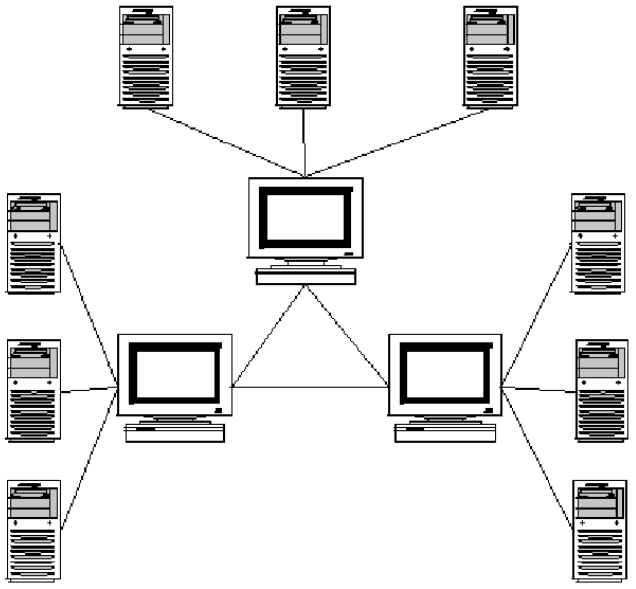

图 2. 半分布式拓扑结构构架图

### 3.2 任务分配的优化

针对任务分配的优化，我们可以采用二级哈希映射算法来分配任务。在进行二级哈希算法的设计之前，我们进行一些假设。首先，我们假设在该系统中的所有 Slave 几点都具有相似的性能，方便我们之后的计算。同时，我们认为可以通过计算增加或减少节点对系统产生的影响，来分析我们的算法是否可以基本实现系统动态可配置性。除此之外，我们还通过查阅资料了解了动态哈希函数应该满足的条件 [3]：

1、平衡性

平衡性是指系统保证每个节点的性能都得到合理利用的性能。本文中系统为了保证平衡性，将所有任务根据 Slave 节点性能尽可能合理的分配到所有 Slave 节点中去。鉴于我们假设每个 Slave 节点处理能力大致相似，我们还应该能保证每个节点承担的任务量应该尽量均衡，否则部分节点将被分配到其性能无法负担的大量任务，而另一部分节点则被分配到极少量任务而造成其性能的浪费。

2、单调性

单调性是系统保证当任务分配完成后有 Slave 节点退出时，任务分配算法不再重新分配针对其他 Slave 节点已经分配好的任务，只重新分配原先分配给退出的 Slave 节点的任务。本文中系统为了保证单调性，采用了较为复杂的二重哈希映射算法。因为简单的哈希映射算法往往不能满足单调性的要求，简单哈希映射算法会将 Slave 节点数量纳入计算作为变量，当 Slave 节点数量变化时，简单哈希映射算法的结果也会不可避免地随着变量的变化而变化，这就将更新原先计算出的所有计算结果。事实上，如果只有少量 Slave 节点退出系统，我们并不需要更新原先的全部任务分配，只需在原先任务分配的基础上重新分配原先分配给退出 Slave 节点的任务即可。假如系统不满足单调性，在大量 Slave 节点加入或退出时，系统将频繁更新任务分配，消耗大量资源。

3、一致性

一致性是系统保证在分布式环境中，每个 Slave 节点都有相同的任务分配信息。如果系统不满足一致性，有可能已分配给一个 Slave 节点的任务被重新分配给了新的 Slave 节点，造成任务分配的混乱。这种情况将极大的降低系统的运行效率。一个优秀的任务分配算法应该保证系统的一致性。

#### 3.2.1 实现任务动态分配的两种算法

假设系统内最多可以被用作 Slave 节点的主机数（整个系统内可能存在的最大 Slave 节点数）为 a，目前可用 Slave 节点数为 b。

（1）一级哈希映射算法

一级哈希映射算法是一种基本映射方法，它只是简单的将新获得的 url 用哈希算法进行数学转换，然后将经过转换后的 url 根据目前可用的 Slave 节点数量、性能等因素进行分配，每个 Slave 节点只需爬取分配给它的 url 即可。用数学公式表达该分配算法的函数就是：

id = hash(url)% node_num（公式 3.1）

公式中 id 是该 url 所分配到的 Slave 节点的编号，node_num 是所有 Slave 节点的总数。

首先，我们用哈希算法将新获得的 url 进行数学转化，然后我们将数学转化后的结果对 b 取模，得到相应 url 分配到的 Slave 节点的编号。可以明显看出，该算法满足上文中提到的平衡性要求，每个节点都得到了合理的利用。然而，该算法并不满足我们在上文中提到的单调性的要求。具体来讲，在增加一个节点的情况下，我们依旧按上述方法将新获得的 url 进行数学转化，然后我们将数学转化后的结果对 b+1 取模，此时依然可以保证系统平衡性，但是将数学转化后的结果对 b+1 取模导致原先系统内大量 Slave 节点编号（对 b 取模）都被重新分配，这进一步导致了资源的大量消耗以及任务分配的混乱，而这使系统不符合单调性；在减少一个节点的情况下该系统也同样不符合单调性。为了改进该算法的弊端，我们提出了二级哈希映射算法 [6]。

（2）二级哈希映射算法

该算法采用一张虚拟节点表，一张实际节点表，结合两张表来分配 url 给 Slave 节点。该算法的思路是首先用哈希算法将新获得的 url 进行数学转化，将数学转化后的结果映射到一张虚拟节点表的编号上，然后将虚拟节点表中的虚拟节点编号映射到实际存在的 Slave 节点编号上。理论上，我们至少需要一次，至多需要两次哈希映射运算来实现二级哈希映射算法。这样可以同时保证系统在保证分布式 URL 分配的平衡性与单调性。基于上述优势，本文中基于 Scrapy 技术的分布式爬虫的任务分配也将使用二级哈希算法。下面本文将具体介绍该算法在本文中系统的应用。

该算法中，我们假设了最多可以被用作 Slave 节点的主机数为 a，而系统当前运行的节点数目是 node_num。相应的每个节点有两张表：一张是虚拟节点表，虚拟节点表假设系统在最大节点数的情况下运行，因此其中有 a 个元素；另一张是实际节点表，实际节点表表示系统目前的运行状况，故表中总共有 node_num 个元素。我们建立一个存储 boolean 类型的数组 Array_max[a] 表示虚拟节点表，若对应相同编号的元素在虚拟节点表和实际节点表中均存在，那么该元素的值为 true，否则为 false；我们建立一个 int 类型的数组 Array_node[node_num] 储存实际节点表，每个元素对应实际 Slave 节点的 ID。

假设新发现的 URL 经第一次哈希映射算法数学转化得到的值为 temp；假设新发现的 URL 为 new_url，我们有 temp =hash( new_url )% a。如果 temp 元素有对应的实际 Slave 节点，那么其值为 true。对于在虚拟节点表中值为 true 的 temp 元素，temp 值就是它所对应的实际 Slave 节点的 ID；对于值为 false 的 temp 元素，该元素所对应的节点并没有实际在现在的系统中工作，此时我们需要将这些 url 分配到实际正在工作的节点上。我们使用第二次哈希映射运算进行本次分配。我们将第二次哈希映射算法所计算出来的值直接对应实际节点数组，其在 Array_node 数组中对应的元素就是实际 Slave 节点的 ID。假设经过第一次哈希映射算法后剩余的 url 为 second_url，我们有 ID = Array_node [ hash( second_url )% node_num ]。通过上述公式我们可以将本来不对应任何实际正在工作 Slave 节点的 second_url 分配给实际正在工作的 Slave 节点，式中 ID 为该节点的编号。这样系统就可以根据计算得到的实际 Slave 节点的 ID 号来传递 second_url，从而达到对现在正在工作的 Slave 节点分配所有任务并且保证系统平衡性，单调性，一致性的目的。整个过程如图 3 所示。

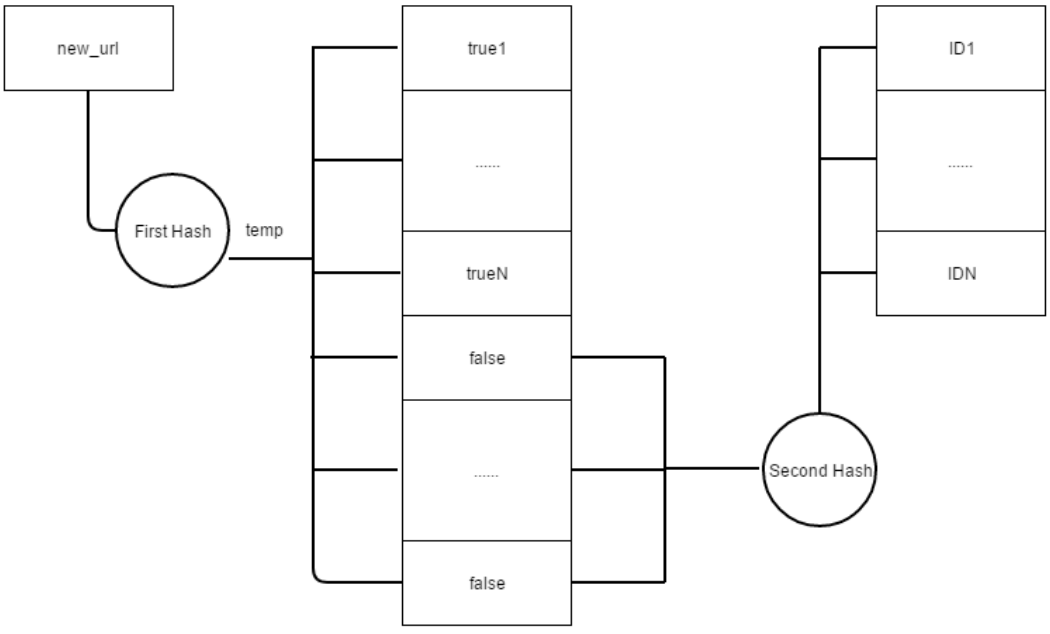


图 3. 二级哈希映射算法

## 4 网络爬虫模块的设计与优化

上文中提到过，我们可以粗略的将分布式网络爬虫分为两个模块，一个是分布式模块，一个是网络爬虫模块。上文中的系统也初步完成了对分布式模块的设计与优化，本章节将重点介绍网络爬虫模块并针对一些情况对其进行优化。

### 4.1 爬虫模块策略设计

本小节将重点从系统工作流程详细介绍该系统爬虫模块策略设计。本系统中，网络爬虫从起点网址进行广度优先遍历，将含有有效信息的 url 都加入到待下载队列当中，对这些 url 进行去重与增量爬取，同时对防爬虫网站屏蔽进行优化。

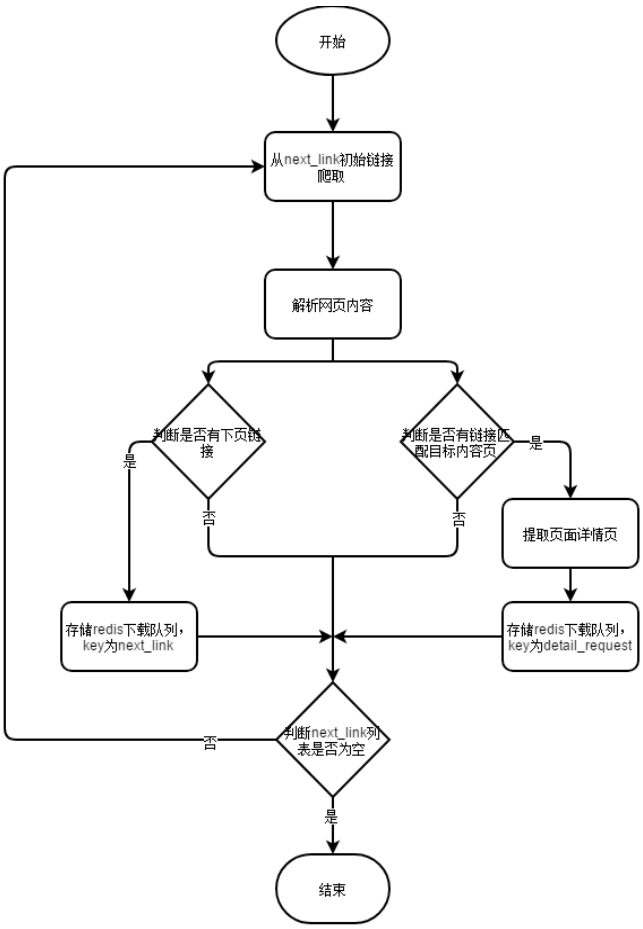


图 4. 爬虫爬取策略

为了实现广度优先遍历，我们需要两种链接，一种是目录跳转页链接，这种链接一般是网站中的下一页链接，使我们能够遍历更深一层的网页中的信息；另一种是目标内容页链接，这种链接都指向我们实际需要爬取得内容，解析该链接后的信息将被存储在 MongoDb 数据库中。一个目录跳转页链接中通常能提取到多个目标内容页链接。这些目标内容页链接通常都被存储到 Redis 数据库的下载队列中并由二级哈希映射算法将它们分配给各个 Slave 节点进行爬取与解析，解析后的信息将被存储在 MongoDb 数据库中。整体爬取流程如图 4。

### 4.2 爬虫模块策略优化

#### 4.2.1 去重与增量爬取

去重与增量爬取，可以保证爬虫不抓取相同的信息，大大提高了资源利用率。本系统中实现去重的方法是，在 Redis 数据库中额外设置已爬取队列，在 Slave 节点每次进行爬取之前，先判断爬取内容是否在 Redis 数据库的已爬取的队列当中。如果已存在，则放弃本次请求。过程如图 5 所示。

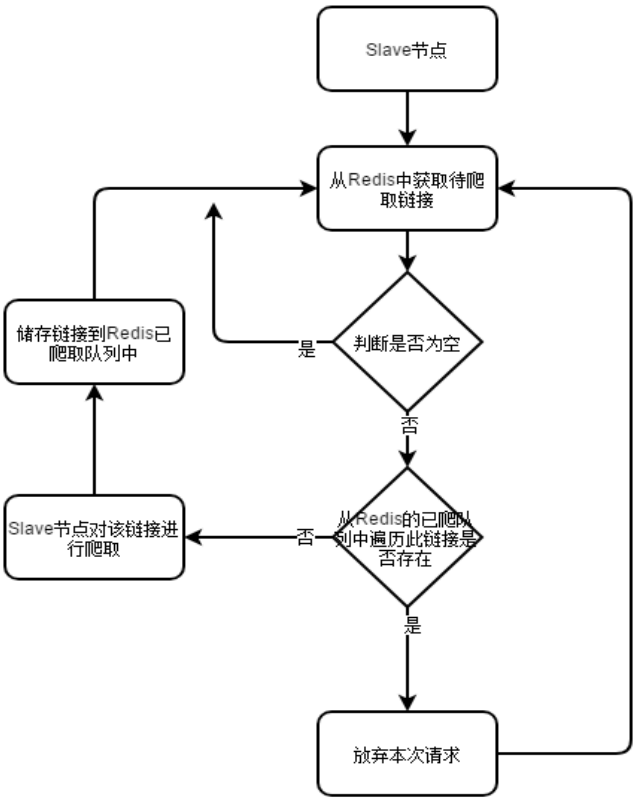


图 5. 去重爬取过程

#### 4.2.2 对防爬虫网站屏蔽的优化

大多数网站都设置了一定程度上的防爬虫策略来避免网络爬虫对其正常运作造成影响。对于同一 ip 的大量快速访问，网站经常认定其为爬虫并通过一些方式来禁止其继续访问，如对该 ip 设置验证码或者剥夺该 ip 的访问权限。然而，本系统的爬虫系统将不可避免的大量快速访问各种网站，如果不适当采取伪装措施，我们可能会被剥夺访问权限而无法维持系统的正常运作。

为此，我们设计并开发了代理 ip 池[7]，以下是 ip 获取步骤介绍，具体流程如图 6 所示。

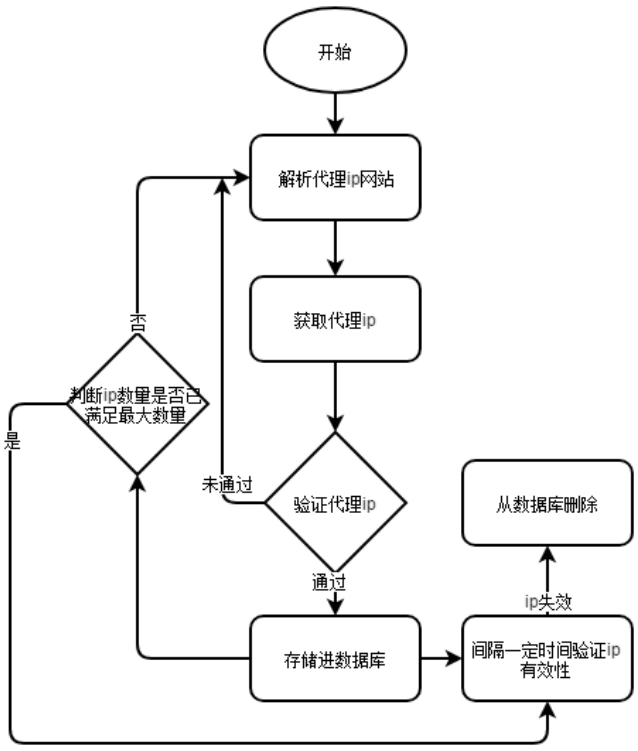


图 6. 代理 ip 池设计流程

1. 解析代理 ip 网站获取大量代理 ip。

2. 验证代理 ip 可用性。

3. 如果代理 ip 可用性验证可行，就将其存储进数据库备用，如验证不可行则放弃该代理 ip，从步骤 1 重新开始。

4. 当数据库中存储的 ip 数量达到最大数量时，停止从代理 ip 网站继续获取 ip；每间隔一段时间对数据库中已存储的 ip 进行可用性验证，保留可用的 ip，删除不可用的 ip，并且获取新的 ip 来填补数据库中 ip 的空缺。

5. 只要有效 ip 数量少于最大数量，就继续获取 ip，重复步骤 1。

## 5 总结

网络爬虫作为当今获取信息的重要手段，也伴随着更新换代与升级。因此，我们需要可以快速，大量，稳定，并且具有可拓展性的分布式爬虫来满足不断膨胀的信息需求。本文初步设计了基于 Scrapy 技术的分布式爬虫，并对其优化与完善提供了思路与方案。

然而，本系统也存在着不足之处：

1) 对系统主从模式的半分布式拓扑结构依然无法完全解决主从结构靠性低和效率低的问题，因为半分布式拓扑结构的性能仍然很大程度上依赖 Master 节点。

2) 对任务分配的优化采用的二级哈希映射算法是在所有 Slave 节点性能相似的假设下进行的，事实上，我们可能面对 Slave 节点性能不同的情况，我们仍需要将 Slave 节点性能作为变量加入我们的优化方法。

我们会在今后的工作中对这些问题进行改进，设计出更完善的优化方案。

**参考文献：**

[1] 库劳里斯(英)分布式系统概念与设计(原书第5版)[M]. 北京：机械工业出版社，2012.

李婷. 分布式爬虫任务调度和 AJAX 页面抓取［D］. 成都电子科技大学硕士学位论文，2015.

[2] 黄志敏，曾学文. 一种基于 Kademlia 的全分布式爬虫集群方法［I］. 计算机科学，2014.3.

[3] 钱建学. 一种基于 Hadoop 的分布式网络爬虫的研究与设计. 2013.12.

[4] http://blog.csdn.net/howtogetout/article/details/51633814

[5]ShaojunZhong. A Web Crawler System Design Based on Distributed Technology. Journal of Networks[J], 2011.12.

[6] 黄礼骏. 分布式系统的升级和数据迁移问题研究. 2015.7.

[7] https://github.com/qiyeboy/IPProxyPool

**【作者简介】**

刘泽华 (1996-)，男，本科，主要研究领域为信息安全，数据科学，人工智能；

赵文琦 (1997-)，男，本科，主要研究领域为人工智能，物联网；

张楠 (1997-)，通讯作者，女，本科，主要研究领域为数据挖掘，无线通讯，互联网应用。

（收稿日期：2017-08-16）