一种基于 Kademlia 的全分布式爬虫集群方法

黄志敏[1,2] 曾学文[2] 陈 君[2]
(中国科学院大学 北京 100049)[1]
(中国科学院声学研究所国家网络新媒体工程技术研究中心 北京 100190)[2]

摘 要 针对将海量爬虫节点组织成全分布式爬虫集群所遇到的高效、均衡、可靠、可拓展等问题,提出了一种基于Kademlia的全分布式爬虫集群方法。该方法通过改进的Kademlia技术建立起爬虫节点间的底层通信机制。在此基础上,根据Kademlia的异或特性及节点的可用资源情况,设计并实现具有任务划分、异常处理、节点加入退出处理及负载均衡的全分布式爬虫集群模型。在实际网络系统上的实验结果表明,该方法能有效利用海量弱计算终端的计算、存储和带宽资源,构建高效、均衡、可靠、可大规模拓展的全分布式爬虫集群。

关键词 Kademlia,分布式爬虫,弱计算终端,海量节点,结构化P2P
中图法分类号 TP301.6 **文献标识码** A

Method for Fully Distributed Crawler Cluster Based on Kademlia

HUANG Zhi-ming[1,2] ZENG Xue-weng[2] CHENG Jun[2]
(Graduate University, Chinese Academy of Sciences, Beijing 100049, China)[1]
(National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China)[2]

Abstract For solving the issues of efficiency, balance, reliability and scalability encountered in organizing a mass of crawler node to form a fully distributed crawler cluster, we proposed a fully distributed crawler cluster method based on kademlia. The method establishes the underlying communication mechanism between crawler nodes by improving the method of kademlia technology. On this basis, we designed and implemented a distributed crawler cluster model with task partitioning, exception handling, node join and exit process and load balance, based on the XOR characteristics in kademlia and available resources of the node. Experiments in the actual system show that this method can take advantages of computing, storage, and bandwidth resources of massive weak terminal to successfully build a fully distributed crawler cluster with efficient, balanced, reliable, and has large-scale development properties.

Keywords Kademlia, Distributed crawler, Weak computing terminal, Massive nodes, Structured P2P

随着互联网时代的高速发展,网络中的信息呈爆炸式增长,传统集中式的爬虫集群已经无法满足大规模互联网搜索引擎的需要,于是基于分布式哈希表(Distributed Hash Table)的全分布式爬虫集群开始渐渐进入人们的视野。分布式哈希表(DHT)是结构化P2P的代表,提供资源位置与资源标识的映射关系,能在有限跳数内准确定位资源的存储位置。Loo等人通过将URL发布到DHT网络中实现爬虫节点间的任务划分[1]。Signh等人提出了基于URL重复和内容重复的双重检测来实现任务去重,同时简单描述了节点加入退出及异常的处理方法,即主要是Seen-URL和Seen-Content的转移[2]。Boldi等人基于Chord环提出了基于虚拟节点的一致性哈希算法,该算法将每个查询分配到目标key值后最近的节点,解决了节点加入退出时资源迁移最小化的问题,同时一定程度上解决了节点的负载均衡问题[3]。Kunpeng等人提出了非DHT的环状结构即任务按节点顺序传递检测的方法,每个节点有多个后继节点供选择,可较好地实现任务划分和负载均衡,但会耗费大量的通信和计算资源[4]。国内关于分布式爬虫的研究主要集中在局域网和广域网领域[5-7],或者是非全分布式结构的爬虫[8-11],在目前的研究结果中几乎没有一个模型能够完全解决全分布式爬虫集群中遇到的可靠性、拓展性、动态性和均衡性等问题。

Kademlia[12]是互联网中应用最广泛的DHT网络协议之一,它为每个节点分配固定大小的唯一标识(NodeID),将网络结构组织成二叉树,所有节点根据ID前缀确定位置,并且是树的叶子节点。Kademlia采用ID异或的值作为距离的度量标准,保证了路由查找的收敛,同时使用K桶构建路由表,每个K桶存储k个逻辑距离在$[2^i, 2^{i+1})$内的邻居节点。

基于上述,本文提出一种全新的基于Kademlia的全分布式爬虫集群方法,用以设计任务划分、异常处理、节点加入退出处理及负载均衡等功能,实现全分布式爬虫集群的设计目

到稿日期:2013-05-06 返修日期:2013-09-02 本文受863重大项目课题:融合网络业务体系的开发(2011AA01A102),中科院先导专项课题:海端交互数据实时处理(XDA6030500),国家科技支撑计划课题:支持增强型搜索的重点新闻网站三屏融合服务(2011BAH11B05)资助。
黄志敏(1989—),男,硕士生,主要研究方向为分布式领域及搜索相关技术,E-mail:huangzm@dsp.ac.cn;曾学文 男,教授,博士生导师,主要研究方向为数字信号处理及数字媒体技术;陈 君 女,副研究员,主要研究方向为网络新媒体。

标：高效、均衡、可靠、可拓展，并通过实际实验验证了方法的可行性。

1 基于 Kademlia 的全分布式爬虫集群模型

1.1 模型结构

本文提出的基于 Kademlia 的全分布式爬虫集群模型结构图如图 1 所示。

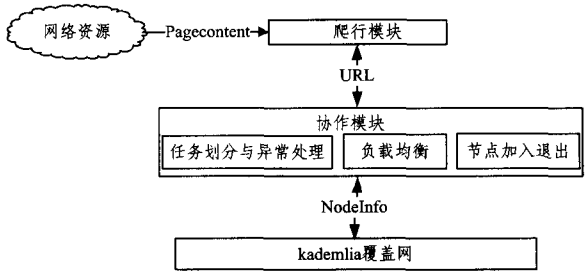


图 1 基于 Kademlia 的模型

其中，爬行模块采用广度优先的爬取方式爬取互联网数据，将解析得到的目标链接转交给协作模块进行处理。协作模块负责在缺少中心控制的情况下，实现节点间的协作机制，即任务划分等功能需求，同时通过底层 Kademlia 覆盖网进行节点间的路由查找和消息通信。

1.2 任务划分与异常处理

基于 DHT 的爬虫集群的任务划分方法一般为采用一致的哈希函数，将 URL 哈希成与节点 ID 相同位数的 key，根据 key 值在覆盖网中查找离 key 值最接近的一个或多个目标节点，将该 URL 发送给目标节点进行处理[1]。由于负责该 URL 的目标节点是唯一的，因此目标节点可以根据保存的已处理的 URL 历史记录实现 URL 的去重，从而使得每个节点各自负责一段 ID 区间的 URL，节点间采用正交爬取，以提升爬取效率。但是，当目标节点出现异常时，属于该节点负责的 URL 区间的 URL 将被分发到其他节点，从而导致 URL 的重复处理，同时如果目标节点在接收 URL 时出现异常，则 URL 将会丢失。

因此，本文提出了一种基于多次哈希与记录备份结合的算法，该算法通过多次哈希选择多个备份节点进行 URL 历史记录备份，将历史记录与目标节点分离开，有效地解决了任务划分与异常处理中遇到的问题，保障了集群的可靠性。算法步骤如下：

(1)选择等待分发的 URL，计算 URL 的二次哈希值为 $h2$，根据 $h2$ 选择备份节点，向备份节点发送 URL 历史记录查询请求；

(2)如果收到备份节点回复该 URL 已处理过，则丢弃该 URL；如果收到备份节点回复该 URL 未处理过，则计算 URL 一次哈希值为 $h1$，根据 $h1$ 选择目标节点，向目标节点发送 URL 分发请求；

(3)如果收到目标节点回复，则转(4)；如果超时未收到回复，则根据 $h1$ 重新选择目标节点并发送 URL 分发请求，重复此步骤直到目标节点成功回复；

(4)根据 $h2$ 选择备份节点，通知备份节点将该 URL 存储到 URL 历史记录中，此次分发结束。

备份节点可根据实际情况进行冗余选择，若节点异常的概率较大，可选择多个备份节点进行 URL 历史记录的备份保存，即可通过多次哈希选择优先级递减的备份节点。当查询时，可按优先级递减逐一查询，当 URL 历史记录更新时，则更新全部的备份节点。

1.3 节点加入退出处理

本文提出一种基于 Kademlia 距离特性的节点加入退出方法，并设计了一种 URL 历史记录存储器，以大大减少资源迁移的计算量，保障集群的可拓展性。对于单纯的爬虫来说，只有 URL 历史记录有必要进行迁移，因此本文讨论的资源迁移只包括 URL 历史记录，其他的资源数据可根据该方法进行相应的改动。

1.3.1 URL 历史记录存储器

Kademlia 中度量节点间的距离采用数学上的异或的方式，导致高位对结果影响较大，异或结果从高位开始出现的第一个 1 将决定 AB 之间的距离在哪个范围区间。由以上特点，本文设计了一种 URL 历史记录存储器，其结构如下：

$$Map\langle int\ firstdiff, List\langle h(URL)\rangle\ history\rangle\ map$$

其中，$firstdiff$ 表示该 URL 的哈希值与节点 ID 按高位递减的方向在第几位开始不同，如图 2 所示。

Bit	9	8	7	6	5	4	3	2	1	0
h(URL)=	1	0	0	0	0	0	0	0	0	0
ID=	1	0	0	1	0	0	0	0	1	1

图 2 firstdiff 示意图

则 $firstdiff=6$，$firstdiff$ 越大，两者的距离就越大。另外，$List\langle h(URL)\rangle history$ 表示 URL 哈希值存储的历史列表。于是，Map[6] 表示所有与节点 ID 从第 6 位开始不同的 URL 历史记录的集合。因此，当节点查询 URL 历史记录存储器来解决 URL 重复问题时，可根据 URL 哈希值与节点 ID 比较得到的 $firstdiff$ 值定位到对应的 $history$，在该 $history$ 中查找将使得查找速度大幅提升。

1.3.2 URL 历史记录转移方法

假设节点 A 刚更新后的路由表中存在的距离最近的 K 桶为第 i 个，则其中的某个邻居节点 B 与 A 在第 i 位开始不同，即 $firstdiff(A,B)=i$，由于 A 上的 URL 历史记录是离 A 节点最近，故其至多从第 i 位开始不同，即 $firstdiff(A, URL)\leqslant i$，则在第 $i+1$ 位，URL 与 B 一定相同，所以比起 A 节点第 $i+1$ 个 K 桶之中的邻居节点，A 节点上的 URL 历史记录离节点 B 更近，故有以下节点动态变化时 URL 历史记录迁移方法：

(1)节点退出

执行步骤如下：

1. 查找 A 节点本地 K 桶，取出非空的 i 值最小的 K 桶，将其中的 n 个邻居节点放入备选集合 $S\{n\}$ 中；

2. 判断 n 值，当 $n=0$ 时，停止；当 $n=1$ 时，所有{map[$i-1$]…map[0]}全部归 $S\{1\}$ 中的节点；当 $n>1$ 时，取 $S\{n\}$ 中的邻居节点，判断 $i-1$ 位与 A 是否相同，相同则放入 $Left\{i-1\}$ 集合，不同则放入 $Right\{i-1\}$ 集合，如图 3 所示。

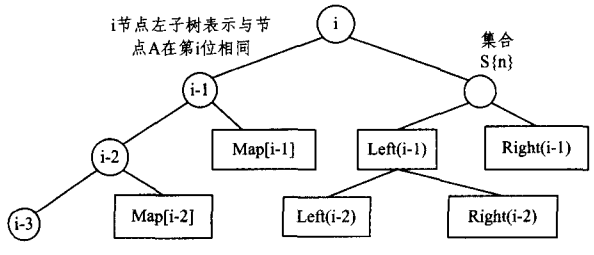


图 3 备选集合分裂图

图3中左右两侧分别表示在第 i 位与 A 节点比较的结果，相同的在左侧，不同的在右侧。

3. 分情况讨论

a) $Right\{i-1\}=0$

$Map[i-1]$ 中的记录逐一匹配 $Left\{i-1\}$ 集合中的节点。

b) $Right\{i-1\}=1$

$Map[i-1]$ 中的记录全部归 $Right\{i-1\}$ 集合中的那个节点。

c) $Right\{i-1\}>1$

$Map[i-1]$ 中的记录逐一匹配 $Right\{i-1\}$ 中的节点。

4. $Left\{i-1\}$ 集合作为 $S\{n\}$ 进入 $i-2$ 层递归，执行2。

递归结束时，A 节点将自身保存的URL历史记录转移到离它最近的K桶中的邻居节点，并根据邻居节点的ID范围进行了划分，使得这些URL历史记录能继续发挥去重的作用。通过利用URL历史记录存储器和异或特性，极大地减少了匹配次数，降低了运算复杂度。

(2)节点加入

新节点 A 加入Kademlia覆盖网时，需要更新自己的路由表，根据前述可知，其应该向最近的K桶中的邻居节点请求属于自己ID区间的URL历史记录。则邻居节点执行的伪代码如下：

```
int fd=FirstDiff(A,B);
for(int i=max;i>fd;i--)
  for(int j=0;j<map[i]. size();j++)
      {
      List(h(URL))第 j 个值为 hj;
      flag=(hj ⊕ ID_B)&(1<<fd)
      if(flag==1)
          MoveToA(hj);
      }
  MoveToA(map[fd]);
```

即节点 B 将URL历史记录存储器中从 $[fd,\max]$ 列表中存储的历史记录哈希值与节点自身ID在第 fd 位进行异或比较，如果结果为1，则该记录的第 fd 位与 A 相同，离节点 A 近，那么需要转移到节点 A，否则不需要转移。$Map[fd]$ 中的所有记录都应该转移到节点 A，因为在第 fd 位上其都与 B 不同，与 A 相同。剩余的 $[fd-1,0]$ 的列表由于离 B 较近，无需转移到 A。同时，由于这些URL离节点 B 最近，其 $first\text{-}diff$ 值不会很小，则 $[0,fd-1]$ 中的记录数量应该要多于 $[fd,\max]$ 中的数量，相应地进行转移时，计算量相比全部的记录数量就小多了。

1.4 负载均衡

全分布式爬虫集群中的每个节点由于硬件设施的差异，导致每个节点的可用资源不同，能承受的工作负载也不同。同时，由于每个节点负责一段ID空间的任务，实际情况中可能遇到在短时间内一节点的任务量急剧增加，导致该节点超负荷运行，而其他节点可能处于空闲状态。因此，适当的负载均衡策略有助于增加分布式爬虫的下载速度和节点利用率。文献[1-3]中并没有清晰可操作的负载均衡策略，而文献[4]中的策略并非基于DHT结构。对于DHT网络中的节点负载均衡策略，有关学者做了大量的研究：Rao等人提出虚拟服务器的方法，将实际节点虚拟成多个互相独立的虚拟节点，节点间进行负载的迁移[13]；David等人使用多个哈希函数虚拟节点的位置，其可以根据负载情况进行动态变化，尽可能地使ID空间均匀分布[14]；Rieche等人通过类似热扩散的物理原理，通过划分ID区间，在每个区间设置冗余的多个节点，将负载在邻居节点间进行转移平衡[15]。

结合Kademlia的路由表特点，考虑节点资源差异，本文提出了一种基于负载等级的负载均衡策略。

1.4.1 负载等级

本文提出的负载等级，是基于工作区和重载块来计算的。其中，工作区用 W 来表示，是节点需要爬取的URL的存储空间，其大小可根据节点自身的计算和存储能力设置；重载块用 H 来表示，是节点间进行负载转移的数据结构，其大小由负载均衡策略统一设置。工作区的大小必须是重载块的整数倍，即 $W/H=n$，n 为整数。当节点接收到其他节点分发的属于自己ID区间的URL时，就将URL存放在工作区，如果工作区已满，则暂时存放在缓存池中，当缓存池中的URL数量满足一个重载块大小时，节点就将缓存池中的数据打包生成一个重载块存储起来。同理，如果工作区有空闲区间可容纳整个重载块，则重载块中的数据将被放入工作区中，并删除该重载块。

综上所述，节点的负载等级定义为：节点已有的重载块数目 S 减去当前工作区可容纳的重载块数目，即当前的工作区空闲的存储空间 W_f 除以重载块的大小 H，即 $L=S-W_f/H$。因此，节点轻载时，负载等级为负，重载时，负载等级为正。轻载等级最小不会超过 $-W/H$，重载等级没有限制。

1.4.2 基于负载等级的负载均衡策略

Kademlia的路由表是根据节点距离，由K桶构造成的。每个K桶存储与自身距离在区间 $[2^i,2^{i+1})$ 范围内的邻居节点信息，即节点路由表中的邻居节点可根据距离划分成为一个又一个的区域，邻近的区域邻居节点数多，越远的区域邻居节点数越少。由于节点只知道局部的节点负载情况，不知道全局的情况，因此本文采取类似热扩散的原理，重载块在邻居节点区域间转移，逐渐达到整体的平衡与稳定。具体的负载均衡策略如下。

首先，每个节点在加入爬虫集群时，需要向启动节点获取重载块大小 H 及更新间隔 T 和下一次更新时间点，根据重载块的大小和自身的负载能力，设置自己工作区的大小 W，需要保证 W 是 H 的整数倍 n。假设URL被分发到达节点服从参数为 λ 的Poission分布，则到达节点的URL总数为Poission分布的均值 $E(N(t))=\lambda t$，所以当更新间隔为 T 时，一次更新间隔到达节点的URL总数为 λT。假设节点爬取时每秒可以抓取 v 条的URL，且对于该节点的URL到达速率有 $\lambda<v$，则可得到节点应该设置的工作区大小：$nH+\lambda T<vT\Rightarrow n<(v-\lambda)T/H$，所以当 n 设置成上述值时，节点可在一个更新间隔内将到达的所有URL及其他节点可能转移给自己的重载块全部都处理掉并保持节点轻载。如果 $v<\lambda$，则节点必然将逐渐囤积URL成为重载节点，为保证URL能及时得到处理且节点负载均衡，则有 $(n+1)H+vT<\lambda T\Rightarrow n<(\lambda-v)T/H-1$，$n>0$，即节点在一个更新间隔内将工作区存满且生成一个重载块，则节点重载，将进行转移，URL将被转移到轻载节点进行及时的处理。如果对节点负载均衡及URL及时性有较高要求，可适当增加 n 的取值，使得上述处理在多个更新间隔后实现，从而增加产量，提高稳定性。

然后，节点在运行过程中，需要定时地更新自己的负载等

级，更新的方法即为计算自己当前的负载等级。节点的第一次更新时间为获取到的下一次更新时间点，以后每次按统一的更新间隔更新。这样可以保证所有节点在负载均衡策略执行的时刻都拥有最新的负载等级。节点更新完自己的负载等级后，如果自己处于重载，则更新路由表中邻居节点的负载等级，否则不更新邻居节点负载等级，同时，不执行最后的负载转移。

最后，重载节点需要在路由表中选择一个合适的轻载节点进行重载块的转移。选取方法如下。

按照路由表中 K 桶的距离范围，从近到远，依次计算路由表中 K 桶的负载等级之和，即该 K 桶中所有邻居节点的最新负载等级之和 L_k。再计算 L_k 与自身负载等级 L 之和，如果 $L_k+L<0$，则说明此区域总体上处于轻载状态，可以容纳该节点转移的重载块数量，再从该 K 桶中选取一个轻载节点进行负载转移，选择方法可有多种，本文采用最简单的方法：选取 K 桶中第一个轻载节点，其负载等级加上节点自身负载等级 L 会小于预设的阈值。当确定负载转移的轻载节点后，就可以将自己的重载块转移到轻载节点。如果 $L_k+L>0$，则该区域总体上处于重载状态，跳过该 K 桶，继续下一个 K 桶的计算。

2 实验与仿真

本节讨论基于 Kademlia 的全分布式爬虫集群方法的实验结果。实验设备选择了分布在全国各地的 8 个实际节点进行算法测试。节点情况如表 1 所列。

表 1 实验节点信息

序号	地理位置	节点 IP
1	声学所	210.75.225.1xx
2	声学所	210.75.225.1xx
3	声学所	210.75.225.1xx
4	北京上地	223.202.17.1x
5	中国科大	120.193.102.6x
6	央视(三元桥)	115.182.34.4x
7	昆山有线	221.6.85.1x
8	国家图书馆	202.106.125.4x

实验中，通过系统的产量、负载均衡及开销这 3 方面来对本文提出的全分布式爬虫集群方法的性能进行评估。

2.1 产量

设计实验 1 如下：设置每个节点具有相同的爬取速度(每隔 10s 处理一个 URL)，取 3 分钟为采样间隔，初始系统中只有 2 个节点，随后每隔 5 个采样周期增加 2 个节点，第 15 到 20 个周期增加 1 个节点，最后在保证节点的待爬取 URL 充足的情况下，可得图 4 所示。

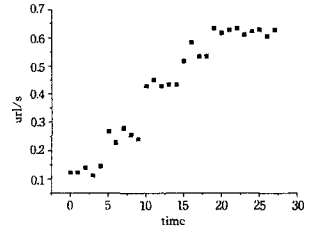


图 4 系统下载速度

由图 4 可知，系统的下载速度在 5 个采样周期内基本保持稳定，当有节点加入时，下载速度有明显提升，最后保持在稳定的水平上。计算在不同节点数量下的所有采样数据的平均值，如图 5 所示。

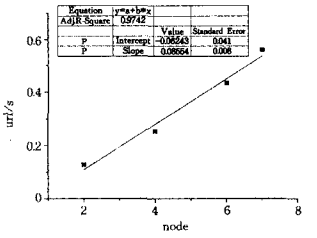


图 5 系统节点拓展图

由图 5 可知，系统的爬取速度随着节点数量的增加呈线性增长，这意味着系统的可拓展性良好。为了增加产量，可以按比例增加节点数量。

2.2 负载均衡

设计实验 2 如下：首先，将节点分成 3 类，分别是爬取速率 $v=0.1$，$v=0.2$，$v=0.05$，设置重载块 $H=20$，每个节点工作区大小统一为 $W=100$，记录在未采用负载均衡策略下的统计数据，计算各节点的 URL 到达速率 λ；然后，根据本文提出的节点工作区大小计算公式，结合实际情况，设置节点的工作区大小，如表 2 所列。

表 2 实验节点参数设置

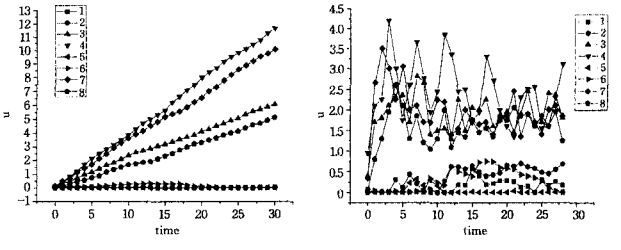
节点序号	爬取速率 v	URL 到达速率 λ	工作区大小 W (倍数 n)
1	0.2	0.083	120(6)
2	0.2	0.08	120(6)
3	0.05	0.16	20(1)
4	0.1	0.34	20(1)
5	0.2	0.09	120(6)
6	0.1	0.083	80(4)
7	0.1	0.22	20(1)
8	0.05	0.136	20(1)

最后，在上述的参数设置下，采用本文的负载均衡策略重做实验，记录系统的负载情况，并与未采用的情况进行比较。本文采用负载率来比较节点的负载情况，具体计算公式如下：

负载率(u)＝待爬取 URL 数量/工作区大小

即节点目前的负载程度为其目前承担的工作量除以其最大可承受的工作量，可见，u 维持在 1.0 左右为最佳情况。

如图 6(a)所示，在未采用负载均衡策略时，如果节点的到达速率大于节点的爬取速率，则节点将一直处于重载且负担越来越重，而有些节点一直处于轻载状态，大量计算和带宽资源浪费。如图 6(b)所示，采用基于负载等级的负载均衡策略后，重载节点负载率都大大降低，轻载节点负载率也有所提高，且并未出现轻载节点突变成重载节点的情况，说明基于负载等级且考虑区域信息的策略可以使系统整体负载率渐渐达到最优，而不会出现局部的负载剧变。



(a) 未采用负载均衡策略　　(b) 采用负载均衡策略

图 6 节点负载率对比

如图 7 所示，在采用负载均衡策略后，系统的产量有最大 33.3%的提升，且随着时间的推移有明显的上升趋势。随着时间推移，重载节点负载越来越重，累积的任务增加但产出却是固定的，在采用负载均衡策略的情况下，轻载节点的产出将逐步提升直到最大产出为止。当节点数增加到几十上百个时，该负载均衡策略的效果更明显。

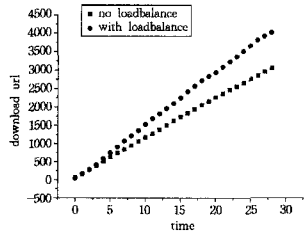


图 7　系统产量对比

图 8 显示了负载均衡策略的采用对系统通信开销的影响。可见负载均衡策略中的重载块转移对系统通信开销影响并不明显。由于 Kademlia 在维护路由表及路由查找时需要进行大量的节点间通信，因此不同网络情况下节点间的路由通信开销才是主要的。节点重载块的转移只在邻近区域选择单一节点进行，所需的通信开销是固定的，当节点大规模拓展时，节点间的路由通信开销将大大增加，使得负载均衡策略带来的通信开销影响更小了。

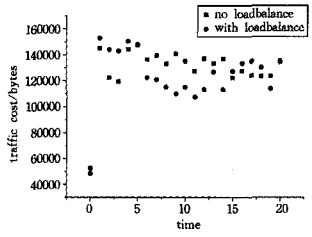


图 8　通信开销对比

2.3　系统开销

表 3 记录了在实验 2 的基础上节点的资源使用率(括号内为节点未部署分布式爬虫时的情况)，在不同的爬取速率下，每个节点的 CPU 带宽使用率以及内存占有率都有不同程度的提升，在利用节点空闲资源的同时不影响节点的正常工作，在可接受的资源使用率范围内可以提升爬取速率，增加系统产量，增大节点资源使用率。

表 3　实验节点资源使用率

序号	%CPU	%内存	%带宽
1	33.7(12)	1.7(1.0)	15.5(2.1)
2	36(13.2)	1.8(0.9)	18.3(2.3)
3	15.3(10.8)	1.7(0.8)	5.3(2.1)
4	22.5(12.2)	1.8(0.9)	10.6(2.5)
5	34(11.4)	1.6(0.9)	16.0(1.9)
6	23.7(10.1)	1.8(0.8)	10.7(2.6)
7	22.9(11.1)	1.7(0.8)	11.1(2.2)
8	18.2(12.5)	1.7(0.9)	6.2(2.1)

结束语　通过实际实验，本文提出的基于 Kademlia 的全分布式爬虫集群方法能在保证爬虫集群正常工作的前提下，有效提高系统负载均衡能力，增加系统产量，实现系统良好的可靠性和可拓展性。同时，本文提出的 Kademlia 环境下的节点加入退出算法及负载均衡算法具有广泛的应用前景，下一步将继续优化方法，同时将本方法应用到其他领域。

参 考 文 献

[1] Loo B T, Cooper O, Krishnamurthy S. Distributed web crawling over DHTs[R]. University of California, Berkeley, 2004

[2] Singh A, et al. Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web Distributed Multimedia Information Retrieval[J]. Distributed Multimedia Information Retrieval(Lecture Notes in Computer Science), 2004, 2924: 126-142

[3] Boldi P, et al. UbiCrawler: a scalable fully distributed Web crawler[J]. Software: Practice and Experience, 2004, 34(8): 711-726

[4] Zhu K, et al. A Full Distributed Web Crawler Based on Structured NetworkInformation Retrieval Technology[J]. Information Retrieval Technology(Lecture Notes in Computer Science), 2008, 4993: 478-483

[5] 许笑，张伟哲，张宏莉，等. 广域网分布式 Web 爬虫[J]. Journal of Software, 2010, 21(5): 1067-1082

[6] 吴黎兵，柯亚林，何炎祥，等. 分布式网络爬虫的设计与实现[J]. 计算机应用与软件，2011, 28(11): 176-179

[7] 刘爽，姜春祥，张伟哲，等. 基于 GNP 算法的分布式爬虫调度策略[J]. 计算机应用研究，2010(2): 446-449

[8] 袁理锋. 分布式视频搜索爬虫系统的设计与实现[D]. 大连：大连理工大学，2009

[9] 李伟. 分布式搜索引擎设计与实现[D]. 安徽：中国科学技术大学，2006

[10] 金凡，顾进广. 一种改进的 T-Spider 分布式爬虫[J]. 微电子学与计算机，2011, 28(8): 102-104

[11] 中国科学院声学研究所. 一种网页爬虫协作方法：中国，CN201110375264.1[P]. 2012-05-30

[12] Maymounkov P, Mazieres D. Kademlia: A peer-to-peer information system based on the xor metric[C]// Peer-to-Peer Systems. 2002: 53-65

[13] Rao A, et al. Load Balancing in Structured P2P Systems [C]// Proc. 2nd Int. Workshop on Peer-to-Peer Systems. Berlin/Heidelberg: Springer, 2003: 68-79

[14] Karger D R, Ruhl M. Simple efficient load balancing algorithms for peer-to-peer systems[C]// Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures 2004. ACM: Barcelona, Spain, 2004: 36-43

[15] Rieche S, Petrak L, Wehrle K. A thermal-dissipation-based approach for balancing data load in distributed hash tables[C]// 29th Annual IEEE International Conference on Local Computer Networks. 2004