A Full Distributed Web Crawler Based on Structured Network

Kunpeng Zhu, Zhiming Xu, Xiaolong Wang, and Yuming Zhao

Intelligent Technology and Natural Language Processing Lab, School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
{kpzhu,xuzm,wangxl,ymzhao}@insun.hit.edu.cn

Abstract. Distributed Web crawlers have recently received more and more attention from researchers. Full decentralized crawler without a centralized managing server seems to be an interesting architectural paradigm for realizing large scale information collecting systems for its scalability, failure resilience and increased autonomy of nodes. This paper provides a novel full distributed Web crawler system which is based on structured network, and a distributed crawling model is developed and applied in it which improves the performance of the system. Some important issues such as assignment of tasks, solution of scalability have been discussed. Finally, an experimental study is used to verify the advantages of system, and the results are comparatively satisfying.

Keywords: Web crawling; full distributed; structured network.

1 Introduction

Due to the exponential growth of the web, an important challenge of web crawler is to efficiently collect massive pages in a limited time frame, one that has received considerable research attention.

Some distributed crawling systems have been worked out to finish Web massive information collecting task. The distributed crawling systems mentioned in [1, 2, 3] use a centralized server to manage the communication and synchronization of crawling nodes. These centralized solutions are known to have problems like link congestion, being a single point of failure, and expensive administration. Some full distributed crawling systems have been proposed in [4, 5, 6], that is, no central coordinator can exist in these systems. In these systems, large numbers of nodes collaborate dynamically in an ad-hoc manner and share information in large-scale distributed environments without centralized coordination. An important issue of the presented crawlers is dynamic load balance. Most systems concern the methods of static load assignment and ignore the unbalance in crawling process. Another issue that has not been well resolved is scalability caused by the arrivals and departures of nodes.

In this paper, our research mainly focuses on how to design a full distributed crawler based on a distributed crawling model. A structured architecture will be proposed and the mechanism to achieve load balance and scalability will be given.

2 Architecture

In our crawler, crawling nodes are organized as a structured ring network to offer the service of collecting Web pages. The ring is composed of several crawling nodes that autonomously coordinate their behaviour in such a way that each of them scans its share of the Web. Such organization has several desirable properties – it is highly resilient to a single point of failure, and incur low overhead at node arrivals and departures. More importantly, they are simple to implement and incur virtually no overhead in topology maintenance. The overview of crawler system can be described by Figure.1.

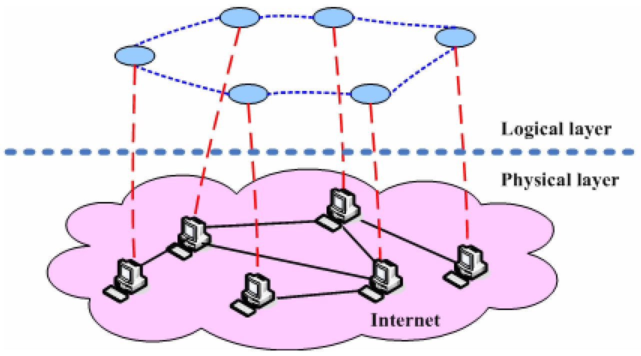

Fig. 1. Overview of crawler system

Each crawling node in system is composed of 2 parts: crawling module and control module. The function of crawling module is to download web pages from Internet according to the URLs queue. The function of control module is to manage the communication and harmony with other crawling nodes.

The inside organization of each crawling node can be described by Figure.2.

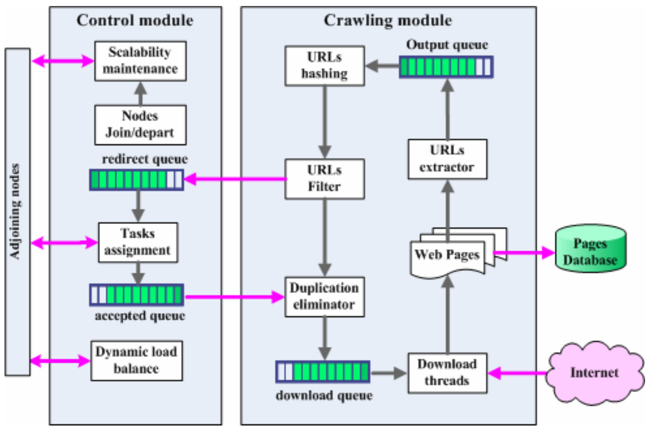

Fig. 2. Architecture of each crawling node

3 Distributed Crawling Model

Our research mainly focuses on the decentralized crawling strategy which has been implemented in control module. The core of the strategy is called distributed crawling model (DCM). As mention above, the model is composed of three parts which will be described as follow.

3.1 Tasks Assignment

The sub-module of tasks assignment is used to divide whole crawling task into different parts, and allocate them to each node in order to achieve a parallel processing. We propose a new method of tasks assignment which is a dynamic consecutive division to the value space of hash function, and we explain why this method makes it possible to decentralize every task and to resolve the above problems.

Let the value space of hash function be a rang from a_0 to a_n. For example, if $H(URL)$ denotes the sum of integer parts of the IP of URL's host, then $a_0 = 0$ and $a_n = 255 \times 4 = 1020$. Let n denote the number of nodes, we can get a division with $n-1$ numbers denoted by $(a_1, a_2, ..., a_{n-1})$ and $a_0 < a_1 < a_2 < ... < a_{n-1} < a_n$, the node i will take charge of the URLs whose $H(URL)$ are located in the range of (a_{i-1}, a_i).

At the beginning, we initialize the value of a_i as follow:

$$a_i = \frac{a_n}{n} \times i \qquad (i = 1,2,...,n-1) \tag{1}$$

Obviously, formula (1) is a n equivalent division on the range of (a_0, a_n). We will dynamically change the value of a_i in crawling process to achieve a load balance, the more detail will described in next section.

The crawling nodes are organized as a ring. Each node has two neighbors which called "preceding-node" and "following-node". The hashing value of URLs on preceding-node is smaller than that on following-node. Each node need maintain three URLs queues: local-queue, preceding-queue and following-queue. The URLs in preceding-queue need to be sent to preceding-node, URLs in following-queue need to be sent to following-node and URLs in local-queue need to be sent to local download queue. In order to complete this process, we define two token in our structured network named "forward-token" and "backward-token". The "forward-token" starts off from the first node in network which charges the set of the smallest hashing value of URLs. The node holding "forward-token" will operate as follow:

1. The node sends its following-queue to its following-node.
2. The following-node will accept the queue and divide the queue into two parts, one is added into its own local download queue, the other is added into its following-queue.
3. The node gives the forward-token to its following-node.

The "backward-token" starts off from the last node and the process is the opposite with the "forward-token".

The time of token walking a circle on the ring is called cycle T. In order to avoid frequent communication in the network, we let T equal a longish time, such as one hour. So, the interval of sending token from one node to another is T/n. And a new URL will arrive at the corresponding node within the time T.

3.2 Dynamic Load Balance Management

Load balance means that each node should be responsible for approximately the same number of URLs. But the n equivalent division on the range of (a_0, a_n) can not assure that there are same number URLs located in each part. So we provide a dynamic load balance model to achieve the characteristic of load balance. Our model is based on three principles:

1. A little unbalance is permitted for the communication price of adjusting load balance.
2. At certain moment, the operation of adjusting load balance only occurs between two adjoining nodes.
3. Local balance should comply with the global.

We use a token called DLBT (dynamic load balance token) to perform the function. The operation of adjusting load balance only occurs between the node holding DLBT and its “following-node”. The DLBT starts off from the first node and walk on the ring.

3.3 Scalability Maintenance

High scalability means that the more crawling nodes, the higher performance. We should develop the mechanism to maintain the topology of structured ring networks and manage the arrivals and departures of crawling nodes.

The mechanism is rather simple. Each node in the structured network not only keeps the information of its two neighbors, but also saves the information of three closest nodes in up and down direction. If a node is failure, its neighbor will find the next node to rebuild the virtual link. If a node joins in the network, it will request for the connected node and get the information of neighbors to create the link, of course, the redundant link will be removed.

4 Evaluation Methodology and Experiment Results

The goal of this section is to analyze the load balance and scalability features of our crawler. In order to achieve the load balance of the system, we use hash function to dynamically assign URLs to each crawling node. The consequence can be obtained by analyzing colleted Web pages by each node every hour.

In Jan 2007, we utilize our crawler to get experimental data which are about 7771402 Web pages with 21.75GB capacity within ten hours. And the number of parsed URLs is about 58606584. All of our measurements are made on six general Intel PCs with the P4 3.0GHZ Intel processors, 2GB of memory and 400GB hard SATA disk, the bandwidth is 100M. The operating system is Redhat Linux 9.0.

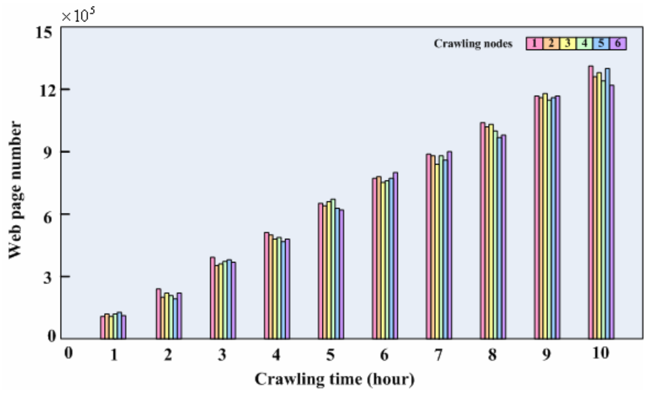

Fig. 3. Evaluation of load balance

Figure.3 shows the performance of load balance of our system. The experimental results show that our dynamic load balance model has a remarkable performance in improving the load balance in distributed crawler systems.

With the more number of crawling nodes, the crawling speed of our system is higher, shown in figure.4. The relation is almost linear. But with the increasing of nodes, the overload of the synchronization and communication among the nodes may decrease the performance.

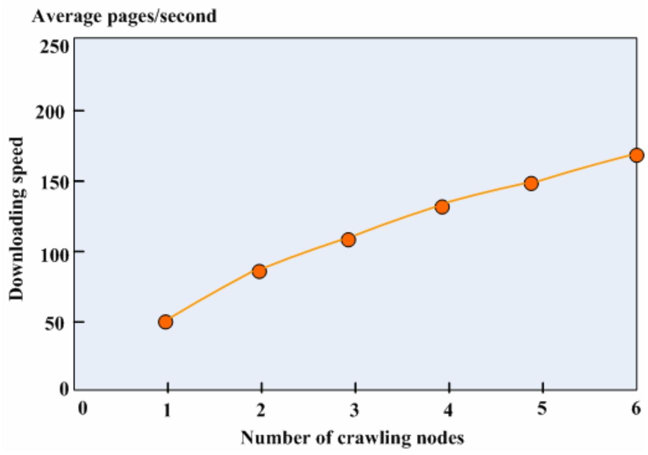

Fig. 4. Evaluation of scalability

5 Conclusions

In this paper we present a full distributed crawler system based on a structured network. A distributed crawling model (DCM) is proposed to achieve the merits of load balance and scalability. And a new method of tasks assignment is presented, which is a dynamic consecutive division on the value space of hash function. Also, a dynamic load balance model is used on the structured ring network. The experiment results show that our methods achieve a well performance to improve the load balance and scalability in distributed crawling environment.

Acknowledgements

This paper is supported by the Key Program of National Natural Science Foundation (No. 60435020).

References

1. Yan, H., Wang, J., Li, X., Guo, L.: Architectural Design and Evaluation of an Efficient Web-crawling System. Journal of System and Software 60(3), 185–193 (2002)
2. Shkapenyuk, V., Suel, T.: Design and Implementation of a High-Performance Distributed Web Crawler. In: Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), pp. 357–368 (2002)
3. Hafri, Y., Djeraba, C.: High performance crawling system. In: Proceedings of the 6th ACM SIGMM international workshop on multimedia information retrieval, pp. 299–306. ACM Press, New York (2004)
4. Boldi, P., Codenotti, B., Santini, M., Vigna, S.: Ubicrawler: A scalable fully distributed Web crawler. Software: Practice & Experience 34(8), 711–726 (2004)
5. Loo, B.T., Cooper, O., Krishnamurthy, S.: Distributed Web Crawling over DHTs. Tech. Rep. UCB//CSD-04-1332, UC Berkeley, Computer Science Division (February 2004)
6. Singh, A., Srivatsa, M., Liu, L., Miller, T.: Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web. In: The Proceedings of the SIGIR workshop on distributed information retrieval (August 2003)