PAPER

Efficient Distributed Web Crawling Utilizing Internet Resources

Xiao XU[†], *Nonmember*, **Weizhe ZHANG**[†a)], *Member*, **Hongli ZHANG**[†], *and* **Binxing FANG**[†], *Nonmembers*

SUMMARY Internet computing is proposed to exploit personal computing resources across the Internet in order to build large-scale Web applications at lower cost. In this paper, a DHT-based distributed Web crawling model based on the concept of Internet computing is proposed. Also, we propose two optimizations to reduce the download time and waiting time of the Web crawling tasks in order to increase the system's throughput and update rate. Based on our contributor-friendly download scheme, the improvement on the download time is achieved by shortening the crawler-crawlee RTTs. In order to accurately estimate the RTTs, a network coordinate system is combined with the underlying DHT. The improvement on the waiting time is achieved by redirecting the incoming crawling tasks to light-loaded crawlers in order to keep the queue on each crawler equally sized. We also propose a simple Web site partition method to split a large Web site into smaller pieces in order to reduce the task granularity. All the methods proposed are evaluated through real Internet tests and simulations showing satisfactory results.

key words: *Internet computing, distributed Web crawling, DHT, network coordinate system, load balancing*

1. Introduction

Web search services are becoming more and more important in everyone's daily life. Their availability and effectiveness largely depend on the efficiency of the underlying crawling systems. The current super large Web search systems such as Google and Yahoo all deploy their crawling systems on well-maintained computer clusters. However, as the size of the internet increases, the clusters have to become larger and larger to cope with the trend thus raising huge maintenance and administration costs.

Based on the inspiration of internet computing [1] and SETI@home [2], a number of large-scale distributed Web crawling (DWC) systems utilizing personal computing resources across the Internet [3]–[5] have emerged lately. By utilizing the large amount of Internet resources, the system can throw the cost of bandwidth, memory space and storage space back to the Internet itself. DWC system faces the following core issues:

Scalability How to maintain high throughput and high stability while the set of crawlers within the system is always changing due to the churns on the Internet.

Efficiency First, how to crawl a Web host as soon as possible while there is no central scheduler in the system. Second, how to reduce the time gap between task submission and execution while the crawlers are all the ordinary PCs with limited capacities and long waiting queues.

There are following DWC systems that have been published in sufficient detail.

Ubicrawler [6] It is the first published system to propose the idea of hash mapping the Web sites to slots (crawlers) in order to partition the set of Web hosts. The system adopts consistent hashing mechanism [7] so that the addition or removal of one slot (crawler) does not significantly change the overall mapping of Web hosts to slots (crawlers). The idea solves the scalability problem that all distributed systems face. And the random generator used in consistent hashing also provides a natural load balancing solution. However, consistent hashing also brings shortcomings, e.g. its hash function is based on a random generator which ignores the information of networking.

Apoidea [8] It made a further improvement to the model of Ubicrawler on scalability. Instead of using consistent hashing, Apoidea adopts a Chord-like DHT to achieve full distribution so that each crawler can independently discover new URLs and routes the URLs if the URLs are mapped to the other crawlers according to DHT's hash-based mapping mechanism. Unfortunately, the system hashes the Web hosts without taking the efficiency issues into consideration. In addition, the lack of central control makes it difficult to derive a global consistent re-crawl strategy.

IPMicra [9] Unlike the above two hash-based systems, IPMicra implements a location-aware mapping mechanism based on an IP address hierarchy imported from infrastructures called the RIRs (Regional Internet Registries). For each Web host, a central coordinator is responsible for finding it an optimal crawler according to the Web host's and the crawlers' IPs. The idea sufficiently reduces the time cost of download process, but it pays few attentions on the scalability issue.

All three systems are all adopting creative system architectures, but they only concentrate on part of the issues we outlined. Inheriting their ideas, our goal is to build a system which combines the advantages of the above three systems while overcomes their shortcomings.

In this paper, we propose a highly scalable DWC system architecture and our solutions to the two efficiency issues. Our system is built on top of a well-known DHT called

Manuscript received December 16, 2009.
Manuscript revised May 28, 2010.
[†]The authors are with Harbin Institute of Technology, China.
a) E-mail: wzzhang@hit.edu.cn
DOI: 10.1587/transinf.E93.D.2747

the Content Addressable Network (CAN) [10]. The crawlers are treated as peers, at the same time the Web hosts are treated as resources maintained by the peers. A manager is built to periodically re-submit all the discovered Web hosts in order to keep them up-to-date.

For the two efficiency issues, we solve the first one by utilizing the Network Coordinate System (NC). Our experiments have shown that the crawlers' performance is greatly affected by the crawler-host RTTs when the crawler's throughput is restricted below the crawler's download bandwidth limit. By shortening the RTTs the crawling process can be sped up. In our system, an application-independent NC is used to estimate the crawler-host "distances". Moreover, by mapping the crawlers' and Web hosts' network coordinates calculated by the NC to IDs and keys of the DHT, we build a "network location-aware" DHT and transforms the task assignment problem to the DHT's space partition problem.

We solve the second efficiency issue by implementing an effective load balancing algorithm. Uneven distribution of loads among the crawlers may prevent the system from updating each target Web site on time. In our experiment we observed that if a crawler is overloaded, a Web host may wait in the crawler's task queue for so long that the wait time exceeds the Web host's pre-configured update interval. We propose an optimization to CAN in order to balance the load among the crawlers. In the method, the overloaded crawlers will forward their crawling tasks to less burdened crawlers. Such load balancing operation is only performed when each Web host is submitted (or re-submitted) into the system. The load balancing method also contributes a lot to the total throughput of the system as it significantly reduces the number of queued crawling tasks.

All the methods are evaluated through a small scale real Internet test and a series of larger scale simulations based on the data collected from the real Internet. All the experiments show satisfactory results.

The rest of the paper is organized as follows: Section 2 presents our system model. Section 3 describes how the system utilizes the network locality in order to speed up the crawling process. Section 4 provides a simple load balancing solution. In Sect. 5, the methods proposed are evaluated through real internet tests and simulations. Section 6 outlines the related works of distributed Web crawling. Finally, in Sect. 7 we conclude the paper and outline the future research directions.

2. System Model

In the following literature, we call each crawler a peer and each Web host a resource. The Web host is the hostname extracted from the URL, for example the hostname of URL http://en.wikipedia.org/BA_model is en.wikipedia.org. A Web site such as wikipedia.org may consist of multiple Web hosts, for example: zh.wikipedia.org, ja.wikipedia.org, fr.wikipedia.org, etc.

As shown in Fig. 1, the system consists of a number of crawlers and a manager. The only task of the manager is to maintain the URL database and submitting crawling tasks to the crawlers. The crawlers all join a DHT, while the Web hosts are inserted to the same DHT. Each crawler's ID is self-calculated under the help of an independent network coordinate service (NC) (the detailed implementation is shown in Sect. 3). The Web host's key is calculated by a randomly selected bootstrap crawler each time when it is submitted by the manager to the DHT (also see Sect. 3). As we are in the context of Internet computing, the power of global control has to be minimized. With the self-organized DHT overlay, the manager doesn't need to monitor the state of the crawling tasks and the behavior of the crawlers.

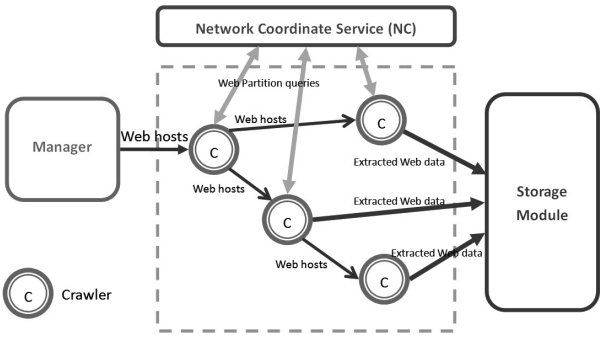


Fig. 1 The architecture of the proposed DWC system.

2.1 Detailed DHT Design

We use the Content Addressable Network (CAN) as the system's underlying DHT. As a result, all the IDs and keys are multi-dimensional coordinates. According to CAN, the whole coordinate space is dynamically partitioned so that each peer is assigned a distinct space called a zone. Each peer holds the information of other peers (we call them the peer's direct neighbors) whose zones are close to its own. The information is used as routing table, in order to route messages between arbitrary peers in the coordinate space.

Upon the joining of each new crawler (peer), firstly the new peer's coordinate is assigned as its ID. Then, According to the JOIN functionality of CAN, after a routing process on the overlay, an already-joined peer's space (or zone) is split into two if the new peer's coordinate finally falls into that peer's zone. The split is done by using a certain ordering of the dimensions in deciding along which dimension a zone is to be split. The new peer picks one of the newly partitioned spaces as its own zone and the already-joined peer continues to hold the remainder space as its zone. The above splitting method proposed by [10] is effective on balanced space partition and zone re-merging. Each peer constantly checks the availability of all its direct neighbors in order to discover the un-notified peer departures. The peer departure algorithm is achieved by merging the departed peer's with other peer's zone using the method proposed in Appendix A of [10].

On the other hand, Web hosts (resources) are submitted by the manager to a randomly selected crawler (as the

bootstrap crawler of the INSERT process). Then they are given coordinates as their keys by the bootstrap crawler. According to the INSERT functionality of CAN, each Web site is assigned to a peer if the Web site's key falls into that peer's zone according to CAN's routing mechanism. When the inserting message of a Web site arrives at its destination peer, the peer invokes the crawling process on itself to crawl that Web site. Different from the case of file sharing, the crawling tasks are difficult to migrate as they consume large system resources. In our system, if a peer A's zone is split by a newly joined peer B, the Web hosts (and the corresponding crawling tasks) within B's new zone (the Web hosts are currently on peer A) don't migrate from peer A to peer B. When a peer leaves the system (or crashes), the crawling tasks running on it are redone by the peer who takes over the peer's zone. The crawling process terminates itself when the crawling task is complete. In order to keep the retrieved Web data up-to-date, all the Web sites will be periodically re-crawled. During each re-crawl, each Web site is re-submitted by the manager to the DHT through a randomly chosen bootstrap peer. Currently the crawlers are designed to work in the firewall mode [11] under which the inter-host Web links are ignored.

According to the algorithm described so far, the relationship between crawlers and Web sites is established in which a peer is responsible for a set of Web sites, at the same time one single Web site is only owned by one single crawler. In addition to this scheme, task replication can be utilized to achieve fault tolerance. But to avoid further complexity, in the following part of the paper, we don't replicate one Web hosts to multiple crawlers.

2.2 The Other Issues

Keeping the crawler's throughput under the contributor's download rate limit is important to our system since all the machines are contributed by the ordinary Internet users who may want to do other things besides crawling (such strategy is called contributor-friendly strategy). In our system, the crawler opens one TCP persistent connections to the target Web host and downloads one Web page (only sends one GET request) each time. The maximum number of concurrent TCP connections (we call the connection limit or CL) is limited to a small value in order to restrict the crawler from occupying too much system resource. If the number of Web hosts (tasks) assigned to a single crawler is over connection limit, the tasks are queued. The queue is a FIFO queue. Each queued Web host (task) will not be executed until one of the executing Web host (task) is done.

If a Web host contains too many pages, a crawler may not be able to traverse all the pages in its life time (since the crawlers are all from the internet users). We have developed a Web host partition method so that each Web host to be crawled is not necessarily submitted to the DHT as a whole at once. Large Web host can be split into smaller pieces according to its sub-directories and the pieces can be submitted to the crawlers in turn. In this context, each piece is treated as an independent task by the crawlers. Our experiment in Sect. 5 shows that using the sub-directories to partition the Web host is feasible. Also, by further reducing the size of the scheduling unit (the unit changes from Web host to piece), the system can achieve a better load balancing.

The Web pages downloaded are not stored on the crawler side. They are sent to a distributed storage system we call the storage module. The crawler removes the structural information of all Web pages, compress the extracted data and insert them to the storage module. We consider this amount of data to be transferred to the storage module is much smaller compared with their original size on the Web. Assuming that the average size of a Web page is 10 KB; typically for a news page the size of the extracted data (content) is 1000 B; after compressing (using gzip) the size of the extracted data usually reduces to less than 1/10 of its original size, then the size of the final data (from a single Web page) to be sent to the Storage Module is less than 100 B, which is 1/100 the size of the original Web page. Moreover, if the crawlers send the data through UDP, the network cost can be further reduced.

3. Efficiency Issue: RTT

3.1 Why RTT

One of our goals is to download the Web data as soon as possible so that we can keep the retrieved Web data up-to-date. However, since our system consists of a large number of low-capacity crawlers, this cannot be achieved by simply increasing the number of TCP connections on each crawler. Instead, in our system, as our crawlers are distributed all over the Internet, each Web host is assigned to a crawler "close" to it on the Internet in order to reduce the download overhead thus increasing the download rate.

When it comes to Web crawling, an intuitive understanding is that the download speed is decided by the bandwidth. Different from this approach, we decide to use the round trip time (RTT) as the metric to estimate the download speed. Under our contributor-friendly download scheme mentioned in 2.2, the actual download time of each Web page consists of three parts (as is demonstrated by Fig. 2):

- The round trip time (RTT) spent on initializing a HTTP request;
- The actual download time (DLT) which is the time gap between the arrival of the first byte of the Web page and the arrival of the last byte of the Web page;
- The politeness wait time (PWT) interval which is a preconfigured constant value.

Note that one Web page may reach the crawler piece by piece as is demonstrated in Fig. 2 because one Web page may be divided into several IP packets. Taking an ADSL connection for example, the typical download bandwidth is 1 Mb/s, the average size of each Web page is 10 KB, then the DLT can be approximately calculated by dividing 10 KB

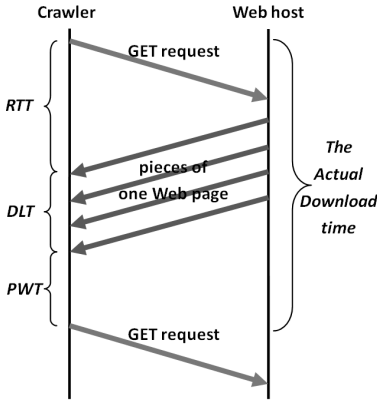


Fig. 2 Download scheme of a crawler.

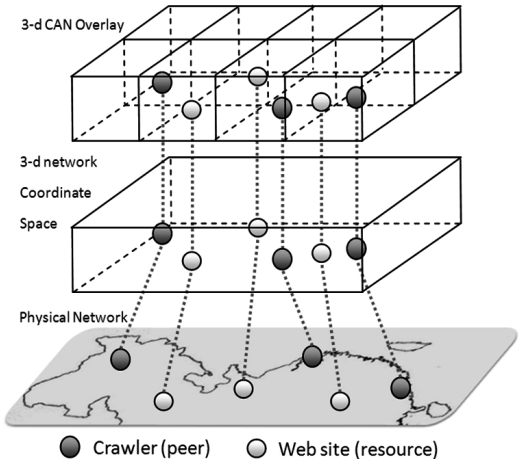


Fig. 3 Mapping from network positions to DHT.

by 1 Mb/s. The result is 80 ms (the actual result may be longer since the bandwidth cannot be completely occupied). In our experience, the typical RTTs are around 50 100 ms, which has the same order of magnitude with the DLT. Assuming that we can save 20 ms of RTT from each Web page, then after downloading 10,000,000 pages (almost the number of pages we use in our experiment), we totally save more than 55 hours. With such amount of time, we can download 2,000,000 more pages (100 ms per page). Actually in our experiment, we observed that the typical RTT/2 reduction under the RTT-optimized method is between 20 ms and 50 ms (which means 40 ms 100 ms of RTT has been reduced).

Another thing to note, by assigning the Web hosts to the crawlers according to the RTTs, the traffic between the crawlers and the Web hosts can be restricted within small network regions (AS or ISP) avoiding a large number of cross-network communications. As a result, to some degree, we also achieve an ISP-friendly system.

It is not easy to collect all the RTTs between arbitrary hosts on the Internet. To solve this problem, we choose to estimate the RTTs using the Network Coordinate System (NC) [12]–[16].

3.2 Combining NC and CAN

The other problem is how to partition the network coordinate space (NC space) in a global consistent manner. We notice that the Content Addressable Network (CAN) provides an effective way to partition a multi-dimensional space. Moreover, as a DHT method, CAN also provides sufficient scalability and stability. Therefore, using CAN to partition the NC space would be a reasonable choice.

3.2.1 3-Tier Mapping Mechanism

The combining of NC and CAN is achieved by adopting a 3-tier mapping. As is demonstrated in Fig. 3, from bottom to top, the 3 tiers are physical tier, coordinate tier and DHT tier.

- On the coordinate tier, a network coordinate service (NC) maps the physical locations (measured using the network latencies) of the crawlers and the Web hosts to the coordinates in a multi-dimensional network coordinate space. The NC is an independent, general purpose Web service that doesn't participate in the crawling process. It is now implemented using the method of GNP [12]. The NC in our system plays a role very similar to the Domain Name System (DNS).
- On the DHT tier, all the crawlers join a DHT (we use the Content Addressable Network) using their network coordinates as IDs, while the Web hosts are inserted to the same DHT using their network coordinates as keys. The DHT's joining and inserting procedure has already been described in 2.1.

The above method can be classified as a proximity identifier selection [17] method used to solve the DHT's topological inconsistency problem. However, different from the existing examples such as Topologically-aware CAN [18] and Proximity-aware CAN [19], our system not only maps the peers (crawlers) from NC to CAN, but also maps the resources (Web hosts or pieces).

3.2.2 Detailed Steps

The first step is to map the nodes on the physical tier to the network coordinates on the coordinate tier. Upon the joining of a crawler, the crawler sends a measuring request to the NC. The NC's landmarks then measure the RTTs to that crawler and send the result and the landmark's coordinate back to that crawler. The crawler then can calculate its network coordinate by itself. When inserting a Web host to the DHT, the Web host should firstly be submitted to a bootstrap crawler. The bootstrap crawler sends a measuring request for the Web host and calculates the Web host's coordinate according to the landmark's answers.

The second step is to map the network coordinates to the DHT tier (CAN space). The network coordinates of the crawlers are used to generate their IDs; on the other hand, the network coordinates of the Web hosts are used to gener-

ate their keys. The number of dimensions in the CAN space is determined by the number of dimensions of the network coordinates.

Each crawler queries the NC for new network coordinate after each time interval. If a crawler's new coordinate shifts too far away from the central coordinate of the crawler's zone (the length of the zone's diagonal is used as the threshold), then the crawler automatically leaves the DHT and re-join using the new coordinate as its ID. During the re-join process, the crawling tasks assigned (submitted) to the crawler still runs. According to Sect. 3, the Web hosts are re-submitted to the DHT during their re-crawl. Before the submission, each Web host's network coordinate is always renewed by the bootstrap crawler, so that they can be assigned to the latest optimal crawler.

3.2.3 The Other Issues

So far, there are still several issues to face. First, many Web hosts reject ping or traceroute probing. According to our experiences, due to the limitation of the gateways (either on the crawler side or the Web host side), more than 60% Web hosts are not reachable by ping and traceroute. Since the access policies of the Web hosts are uncontrollable, the most reliable strategy is to use the http services. As a result, the measured RTTs are actually HTTP-RTTs. We find that although the HTTP-RTTs differ from ping-RTTs (the relative error is about 10%), they can still be estimated accurately by the NC.

Second, some Web hosts have multiple IPs. According to our experiences, approximately 1% Web hosts have multiple IPs. Currently, if a Web host does have multiple IPs we only choose one IP.

Third, the network coordinates have to cope with the fluctuation of the RTTs. Because the RTT between each crawler-host pair is changing during the whole day, the result of a single probing is unable to determine an RTT value with guiding significance. However, the RTT's daily trends subjects to little change. An alternative method is to use the median of multiple RTT test results throughout the day. With the stable RTTs, the network coordinates don't have to change constantly. Even if the network coordinate of a crawler changes, as long as the shift remains smaller than the threshold, it is not necessary to replace the old coordinate with the new one. Finally, the upper bound of the CAN space is different from that of the network coordinate space (NC space). Because the metric of the network coordinate equals to the RTT on the real network, the network coordinate space can be logically infinity. On the other hand, the CAN space should be a limited space with upper and lower bounds on each dimension. In other words, the CAN space is only a subspace of the NC space. Moreover, in order to achieve a balanced load distribution, the node density in the CAN space should be high enough.

According to our analysis to the distribution of nodes on the NC space, most nodes concentrate in a small number of clusters while the rest lies far away from the majority. In our system, we use a simple method (Fig. 4 shows a demonstration on a 2d NC space) to allocate the CAN space from the NCS. The transformation method is under such an assumption that if we have enough favorable Web host samples, we can decide the bounds on each dimension before the system's deployment. In other words, the bounds are trained from well selected samples. Once they are decided, we don't intend to change them while the system is online.

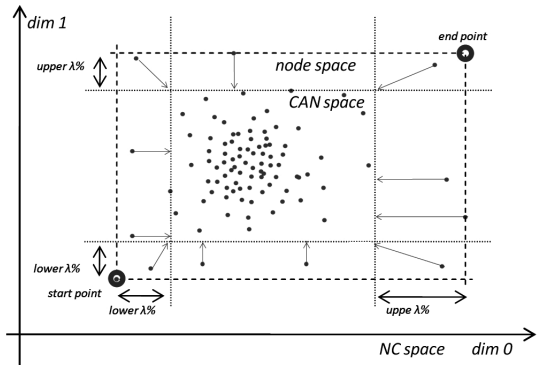


Fig. 4 Demonstration of coordinate transformation (2-dimension).

The samples used to allocate the CAN space is from the historical records of known Web hosts' network coordinates. 1) first, we find the lowest and the highest coordinate values on each dimension. By combining all the lowest coordinate values we get a "lowest" point in the NC space called the start point. Meanwhile, by combining all the highest coordinate values we get a "highest" point called the end point. Now all the nodes are included in the cube space (we call the node space) between the start point and the end point. 2) Then we push the lower bound of the node space on each dimension upward until $\lambda\%$ nodes are left outside. At the same time, we push the upper bound on each dimension downward until $\lambda\%$ nodes are left outside. The cube space within the new bounds are the CAN space we need. 3) Each node left outside during 2) is moved to the boundary closest to it. When a new node (for example a crawler or a Web host) joins the system, its final coordinate can be calculated as follows. Assuming that in the n-dimensional NC space, the lower bounds of the CAN space are $l_0, l_1, \ldots, l_{n-1}$, while the upper bounds are $h_0, h_1, \ldots, h_{n-1}$; the new node's network coordinate given by the NC is $(t_0, t_1, \ldots, t_{n-1})$; then the new node's final coordinate on each dimension is $t'_i = (l_i \le t_i \le h_i ? t_i : (l_i > t_i ? l_i : h_i))$, $0 \le i < n$.

We also tested the other kinds of transformations, for example using arctan (arctan has fixed upper and lower bounds $(-\pi/2, \pi/2)$ to smoothly restrict the bounds of the NC space. However, this method cannot effectively reserve the relative positions between the nodes and makes the scheduling result much worse compared to the method mentioned above. The works of [19] proposes an Affine-transformation-based method to transform the Vivaldi's [16] network coordinates. Under this method, all peers independently adjust their coordinates according to their local knowledge. But the convergence of this method is not

proved. Moreover, we cannot deploy Vivaldi program on all the nodes (the Web hosts are uncontrollable).

4. Efficiency Issue: Load Balancing

4.1 Why Load Balancing

Both conventional hash-based DHT and our new NC-based DHT tend to cause uneven distribution of load among crawlers. The fact has two main impacts to the system:

- Due to the limitation of a crawler's capacity, if a crawler is overloaded, many Web hosts have to be queued, which reduces the system's total throughput and delays the update of the corresponding Web hosts.
- The overloaded crawler process may affect the performance of the residing host causing the contributor's antipathy.

Here we provide a theoretical analysis to the issue. Assuming that the birth of each new Web page on each Web host i obeys a Poisson distribution (mean λ_i). In our system model, each Web host has two states:

Crawling Under this state, the Web host is being crawled by the crawlers;

Wait Under this state, the Web host is either queued by a crawler or is hold by the manager waiting for the next re-submission.

The time cost of the two state are labeled as T_i^{crawl} and T_i^{wait}. The re-crawl interval of Web host i then equals to $\Delta T_i = T_i^{crawl} + T_i^{wait}$. Typically, the interval is a fixed value. We define the Birth-Detect-Delay (BDD) of each URL as $t_{url}^{BDD} = t_{url}^{detect} - t_{url}^{birth}$. The BDD is the time gap between the birth of a new Web page and time when a crawler detects that page. The URLs can be divided into two categories according to their birth time: A) URLs created during T_i^{crawl}; B) URLs created during T_i^{wait}. The number of URLs in category A) equals to $N_i^{crawl} = T_i^{crawl} \cdot \lambda_i$. On the other hand, the number of URLs in category B) equals to $N_i^{wait} = T_i^{wait} \cdot \lambda_i$. Here we use two extreme situations to limit the upper and lower bounds of the BDD. The situations are as follows, assuming that all the URLs of category A) are born at the beginning of T_i^{crawl} and category B) URLs are born at the beginning of T_i^{wait}.

Worst All the URLs are detected and crawled at the end of T_i^{crawl}. This means the BDD of category A) URLs equals to $BDD_{i,A}^{max} = T_i^{crawl}$, while the BDD of category B) URLs equals to $BDD_{i,B}^{max} = T_i^{wait} + T_i^{crawl}$.

Best All the URLs are detected and crawled at the beginning of T_i^{crawl}. This means the BDD of category A) URLs equals to $BDD_{i,A}^{min} = 0$, while the BDD of category B) URLs equals to $BDD_{i,B}^{min} = T_i^{wait}$.

The BDD's expectation in the Worst situation (the upper bound) equals to Equation 4.1.

$$E_{max}(BDD) = \frac{N_i^{crawl}}{N_i^{crawl} + N_i^{wait}} \cdot BDD_{i,A}^{max} + \frac{N_i^{wait}}{N_i^{crawl} + N_i^{wait}} \cdot BDD_{i,B}^{max} = T_i^{crawl} + \frac{(T_i^{wait})^2}{T_i^{crawl} + T_i^{wait}} \quad (1)$$

The BDD's expectation in the Best situation (the lower bound) equals to Equation 4.1.

$$E_{min}(BDD) = \frac{N_i^{crawl}}{N_i^{crawl} + N_i^{wait}} \cdot BDD_{i,A}^{min} + \frac{N_i^{wait}}{N_i^{crawl} + N_i^{wait}} \cdot BDD_{i,B}^{min} = \frac{(T_i^{wait})^2}{T_i^{crawl} + T_i^{wait}} \quad (2)$$

If we assume that T_i^{crawl} has been minimized by utilizing the RTT optimization, then T_i^{crawl} can be treated as a constant value. From Equation 4.1 and 4.1 we can conclude that the lower and upper bounds of BDD linearly depend on the value of T_i^{wait}. And a large portion of T_i^{wait} is the time the Web host spends in the crawler's task queue.

The above analysis shows that if a Web host is queued for too long, the system cannot be able to guarantee high update-rate. The issue becomes more serious when the connection limit of each crawler is a small value (which matches the case of our system model), because long task queues are unavoidable.

Since queuing time cannot be avoided, the only choice is to balance the load among the crawlers and keep the length of each crawler's queue short.

4.2 Load Balancing on INSERT

The transformation method mentioned in Sect. 3 only relieves the load balancing issue but can't substantially solve it. Here we propose a heuristic method called load balancing on INSERT (LBI). Different from the existing load balancing method for CAN which focuses on placing the peers on the CAN space [20]–[24], the method focuses on placing the resources upon their INSERT to the system. The main principle of the method is that if a peer is overloaded, then the resources to be inserted to the peer are forwarded to another peer which is less loaded. In our system, the process of resource insertion equals to the process of Web host submission. Because each Web host is periodically re-submitted (re-crawl) to the DHT, the load balancing operations can be periodically performed.

Assuming that we have a new resource (Web host) to be inserted to the system, first, we have to find the peer (we call it the proto-owner of the resource) in whose zone the resource's key lies according to CAN's routing algorithm. Second, upon the arrival of the resource, the proto-owner compares its relative task load (RTL) with its pre-configured Overloading Threshold (OT). The RTL of a crawler is calculated through the following formula, in which N_{pieces} represents the number of active Web hosts (pieces) assigned to the crawler (including the Web hosts or pieces queued); $BW_{download}$ represents the download bandwidth (Mb/s) that

the crawling process can utilize on the crawler; S_{memory} represents the memory size of the crawler (GB).

$$RTL_i = \frac{N_{pieces}}{min(BW_download, S_{memory})} \quad (3)$$

OT is calculated through Formula 4. N_{pieces}^{max} is a contributor-defined value. Different contributors can configure this value according to their will and their machines' capacity. Note that typically a crawler's OT should be no less than its connection limit (CL). Otherwise, the crawler's actual connection limit should reduce to OT.

$$OT = \frac{N_{pieces}^{max}}{min(BW_download, S_{memory})} \quad (4)$$

We also define the relative page load (RPL) as:

$$RPL_i = \frac{\sum_{i=1}^{i=N_{pieces}} N_i^{pages}}{min(BW_{download}, S_{memory})} \quad (5)$$

N_i^{pages} is the number of pages within each Web host (piece). This number is inherited from the record of the Web host's last crawl.

If the RTL exceeds OT, then the proto-owner considers itself overloaded. In this circumstance, the proto-owner compares its RPL with the RPLs of all its direct neighbors and forwards the message containing the resource (crawling task) to the peer that appears to be the least loaded. The RPLs can be exchanged among neighbors on the overlay by packaging the information in the routine heart beat messages. A TTL (time to live) is added to the forwarded message of the resource, so that the message can be forwarded again by the receiving peer in order to find a peer with even lower load. The TTL is subtracted by 1 after each hop. When the TTL is reduced to 0, the last peer on the forwarding path should unconditionally accept the resource as its own.

We use the RTL to decide whether a crawler is overloaded because using the number of Web hosts (pieces) is relatively easy to conclude a stable OT. On the other hand, we use the RPL to decide the forwarding destination because the RPL shows the work load assigned to the crawler and indicates how soon the forwarded task is to be executed on the destination crawler.

According to the algorithm shown above, there are 2 steps involved in the process of an resource insertion. The steps are demonstrated in Fig. 5.

Routing A user submits an insert request of a resource to a randomly selected bootstrap peer. The bootstrap peer then finds the proto-owner of the resource using the routing algorithm of CAN.

Forwarding Considering itself overloaded, the proto-owner forwards the insert request to a certain lower loaded peer called the real-owner.

The detailed description is outlined in Fig. 6. In the insertion process, firstly, the insert function of a randomly selected bootstrap peer is invoked. The process just follows the traditional routine to route the INSERT request to the proto-owner. On considering itself overloaded, the proto-owner then calls the *insertForward* function of a selected neighbor to forward the insert request. The function *findALowestLoadedNeighbor* within *insertForward* is supposed to return a peer with the lowest load among the local peer's direct neighbors. In the forwarding path, only *insertForward* function NOT *insert* function is called by each peer. The forwarding process only adds a constant (TTL) to the routing cost of CAN ($d\sqrt[d]{N}$). In addition, the time cost on INSERT is very very small compared with the time cost on downloading the Web pages. So the LBI method doesn't bring too much additional cost to the system.

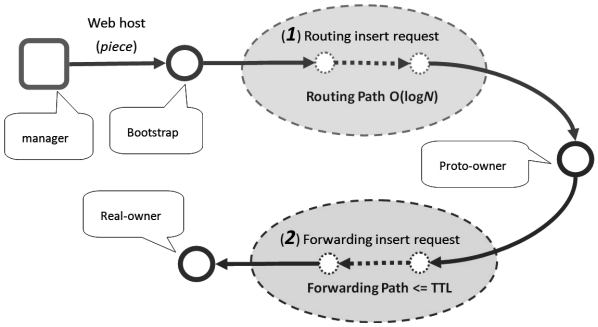


Fig. 5 Steps of an INSERT operation.

```
OT          the overloading threshold
T           the TTL of the forwarded message
class InsertMessage { // the insert message of a resource
    TTL = 0; // time to live
    key; // key of the resource
}
Peer.insert ( InsertMessage request ) {
    if (request.key falls in local zone) {
        if ( RTL >= OT ) {
            next = findALowestLoadedNeighbor();
            if (next == this) {
                resource is accepted by local peer;
                return;
            }
            request.TTL = T;
            next.insertForward(request);
            return;
        }
        resource is accepted by local peer;
    } else {
        next = computeNextHopPeer();
        next .insert (request);
    }
}
```

```
Peer.insertForward (InsertMessage request) {
    request.TTL --;
    if (request.TTL == 0) {
        resource is accepted by local peer;
        return;
    }
    next = findALowestLoadedNeighbor();
    if (next == this) {
        resource is accepted by local peer;
        return;
    }
    next.insertForward(request);
}
```

Fig. 6 The pseudo code of the LBI method.

The forwarding doesn't lead to a global minimum point. It only achieves a limited local-area optimization. However, according to our experiments, this approach is well enough. Another thing to note is that in certain conditions, circles may exist in the forwarding path in which a peer receives the forwarded resource twice. This situation may occur if a peer reduces its load and informs its neighbors in a very short time. We consider the circles no harm to our overall strategy, since our goal is forwarding the resources to less loaded peers no matter whom the peer is.

The LBI method may increase the crawler-host RTTs as many Web hosts are not assigned to their proto-owners. We have implemented more complicated methods taking both RPL and RTT into consideration (shown in 5.3.2). However, in the experiment, we find that these methods don't outperform the LBI. Our experience shows that load balancing performs an important role in deciding the system's throughput. The benefit that the LBI brings exceeds its drawback on RTTs.

5. Experiments and Evaluations

In this section, we conduct three kinds of experiments to prove our theories and evaluate our methods proposed in the previous sections.

5.1 Small Scale Real Internet Test

We conducted a series of experimental crawling on the Chinese Web in 2008. The experiment involves 10 crawlers deployed on four major cities in China Mainland. The target Web hosts are carefully selected 203 large hosts (more than 10000 pages per host) distributed all over China Mainland (at least one Web host in each province). Each crawler's download rate is limited to 2 Mb/s. We keep the number of Web hosts small so that the crawling speed won't exceed the crawler's download rate limit. During the crawling, the crawler opens one TCP persistent connections to the target Web host and downloads one Web page (only sends one GET request) each time. We tested several scheduling schemes:

Random Web hosts are randomly assigned to the crawlers.

City Each Web host is assigned to the crawler which is geographically close to it.

ISP-City Each Web host is assigned to the crawler within the same ISP. If there are multiple choices, we choose the geographically closest crawler.

RTT-worst For each Web host, choose the RTT-farthest crawler.

RTT-best (RTT-top-1) For each Web host, choose the RTT-closest crawler.

NC-CAN The proposed method using GNP as the NC and using the LBI method.

All the RTTs used in the experiment are HTTP-RTTs. During the experiment, we measured the relative error between the HTTP-RTTs and the ping-RTTs. The result is shown in Fig. 8. We can see that 70% HTTP-RTTs have a relative error less than 0.1, which indicates that the HTTP-RTT and the ping-RTT are very similar. Note that the statistics are derived from only 40% of the Web hosts since the rest ones are unreachable by ping.

The average aggregated download rates (the total throughput of the system) of all the 6 methods are shown in Fig. 7. The ordinates are calculated through Eq. (6). N represents the number of Web hosts crawled by the system; D_i represents the total data size downloaded from host i; T_i represents the total download time of host i which is fixed to 15 minutes. We simply add the average throughputs of all Web hosts because they are concurrently crawled. No Web host was queued.

$$Throughput = \sum_{i=1}^{i=N} \frac{D_i}{T_i} \tag{6}$$

From the figure we can see that the RTT performs a key role in deciding the system's total throughput: the method of *RTT-worst* performs the worst, while the method of *RTT-best* performs the best. Note that *RTT-best* is difficult to implement in a fully distributed manner: the only choice is to build the whole scheduling logic into the manager which is inconsistent with our original intention. The figure also shows the performance of the LBI method (*NC-CAN*). Although it doesn't perform as well as the *RTT-best* does, the result is encouraging. Figure 8 shows the relative error of the Web hosts' network coordinates. The figure indicates that although we are using the HTTP-RTTs, the NC still achieves high accuracy. Figure 9 shows the node distribution before and after the transformation proposed in 3.2. The original NC space is an unbounded space and all nodes are

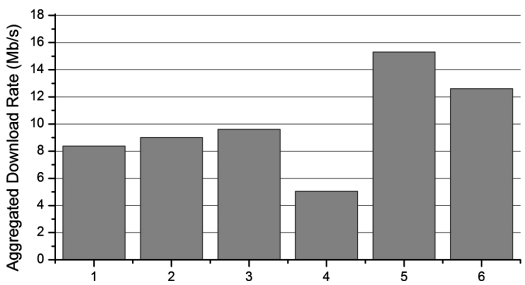


Fig. 7 Throughputs of different scheduling methods.

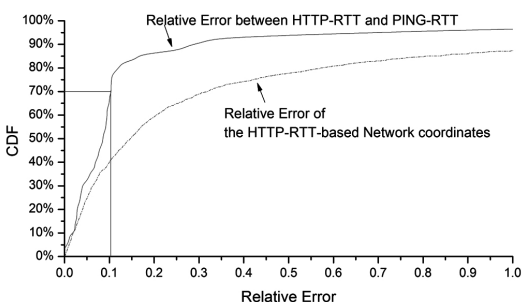
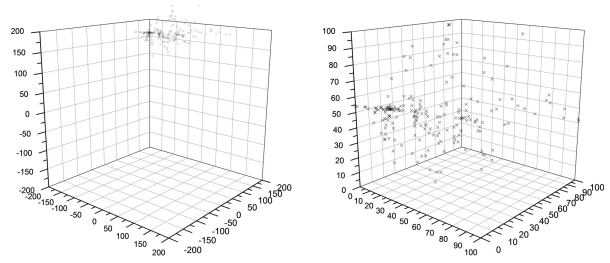


Fig. 8 Relative errors of the HTTP-RTT.



(a) NC space (b) CAN space

Fig. 9 Node distribution before and after transformation.

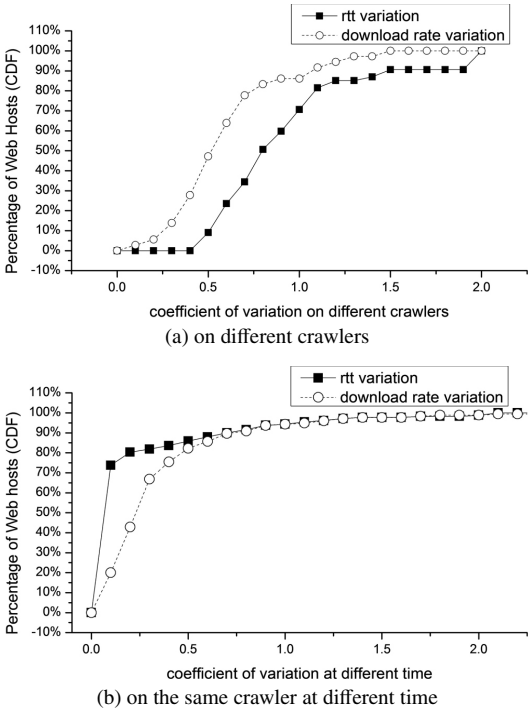
located above the origin of the space. The allocated CAN space is a bounded space and nodes are distributed much more uniformly.

The following facts can explain the RTT's importance. First, the download process involves the RTT. In a typical download scenario, a crawler downloads Web pages from a certain Web host through http persistent connections, so that multiple Web pages can be downloaded in one TCP connection. In order to perform politely, the crawler should only keep one TCP connection to the Web host at one time. After each download, there must be a politeness wait time interval. As is mentioned in 3.1, the total download time (TDT) of a Web page is determined by RTT, DLT and PWT. It is obvious that under fixed DLT and PWT, if the RTT is longer, the total download time will be longer.

Second, RTT has deeper impact to the total download time. For a typical ADSL user with 1 Mb/s download bandwidth, the time cost for downloading 1 page (10 KB) is approximately 80 ms; while a typical RTT is 20 100 ms. This means that the RTT and the DLT have the same order of magnitude. However, the RTT plays a more important role in estimating the total download time. The conclusion is based on the following two observations:

1. Figure 10 (a) is drawn according to the results of 1738 separate crawls. In each crawl, one Web host is crawled separately by the 10 crawlers mentioned above so that 10 different RTTs (the median result of multiple RTT tests throughout the day) and download rates (the median result of multiple download tests) are recorded. We than calculate the coefficient variation of the 10 RTTs and the 10 download rates of each Web host, and put them in the CDF. From Fig. 10 (a) we can see that the RTT subjects to more variation than the download rate (which determines the DLT) on different crawlers.
2. Figure 10 (b) is drawn according to the results of 1738 separate crawls on one crawler. In each crawl, the RTTs and the download rates of a Web host is recorded every 30 minutes throughout 6 hours. We than calculate the coefficient variation of all RTTs and all download rates of each Web host, and put them in CDF. From Fig. 10 (b) we can see that the RTT subjects to smaller variation then the download rate on one crawler.

The first observation shows that the crawler-host RTT



(a) on different crawlers

(b) on the same crawler at different time

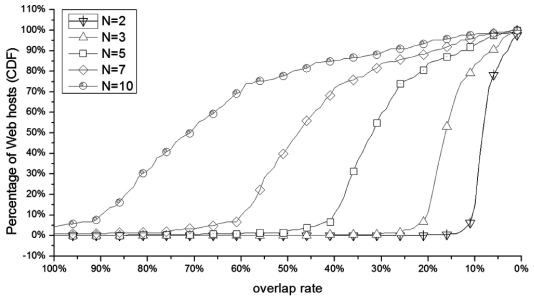
Fig. 10 Stability comparisons between RTT and download rate.

can be dramatically reduced by choosing the right crawler. The second observation shows that it is easier to conclude a stable scheduling strategy from the RTT. The above conclusions makes the RTT a better tool in estimating the total download time. Moreover, the RTTs can be estimated by the NC, which further improves the ease of RTT's utilization.

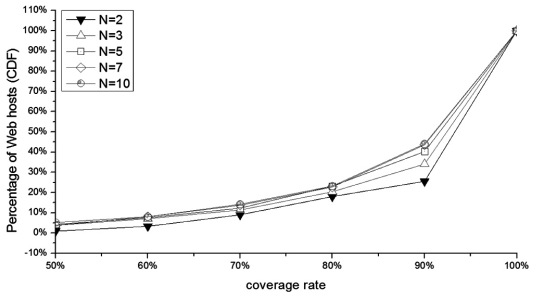
5.2 Splitting Single Web Host into Pieces

In the system, the basic scheduling unit is the Web host. However, some Web hosts are so large that each of them may contain millions of Web pages. Since our crawlers are all contributed by the ordinary Internet users, a crawler may not be able to traverse all the pages on this kind of Web hosts in its life time. We figured out an experimental Web host partition method to split a single large Web host into smaller pieces. These pieces can be separately submitted to the crawlers so that the original Web host can be crawled piece by piece until all the pages have been traversed. In order to minimize the impact of partitioning, the pieces should have small overlap and should cover most of the Web pages on the Web host.

A typical Web host consists of a number of sub-directories. Here we only focus on the sub-directories on the Web host's first level (depth = 1). For each Web host, we find that on average 19% first-level sub-directories (we call the key directories) contain more than 90% Web pages on that host. The result is concluded from our analysis over 1000 Chinese Web hosts (mostly news sites and university sites). To split a Web host into N pieces, we first find out the top K key directories of the Web host. As a result, the Web



(a) overlap rate



(b) coverage rate

Fig. 11 The overlap and coverage caused by the partition method.

host can be partitioned into K + 1 sub-sets. Here 1 (one) refers to the rest less important sub-directories). Then we uniformly assign each sub-set to N pieces and N must be lower than K + 2.

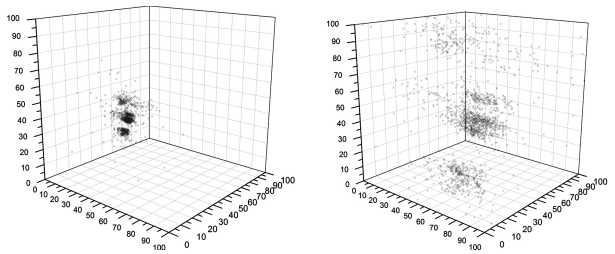
For each sub-set, we use the sub-directory's path as a filter and use the URL of the Web host's main page as the seed URL. If a sub-set (contained in a piece) is submitted to a crawler, the crawler will traverse the Web host from the seed URL and only download the URLs that can pass the filters (since a piece may contain several sub-sets, there may be multiple filters).

Figure 11 shows the overlap rate and coverage rate caused by our partition method under five different Ns. On one hand, we can see that when the number of pieces (N) increases, the overlap rate increases accordingly. The best choice may lie between N = 2 and N = 5. On the other hand, the coverage rates under different Ns subject to little difference. Under the worst condition, only 50% Web pages are found. However there are still 80% Web hosts whose coverage is bigger than 90%.

5.3 DHT Simulations

5.3.1 Simulation Setups

In this sub-section we conduct a series of DHT simulations based on PlanetSim[†] to evaluate the proposed load balancing method. We adopt MIT's King dataset[††] to simulate the RTTs between the crawlers and the Web hosts. We adopt the algorithm of GNP to map the network hosts into coordinates of a 3d coordinate space. The node distribution of the NC space and the CAN space are shown in Fig. 12. Although Fig. 12 (b) shows much better node distribution than



(a) NC space (b) CAN space

Fig. 12 Node distribution before and after transformation (King dataset).

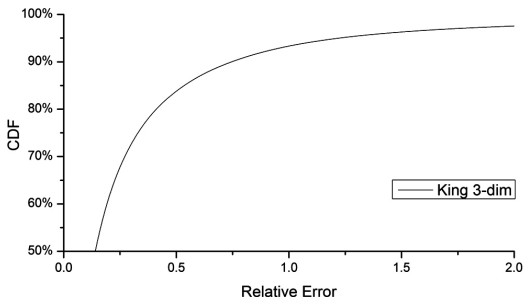


Fig. 13 Relative errors of the network coordinates.

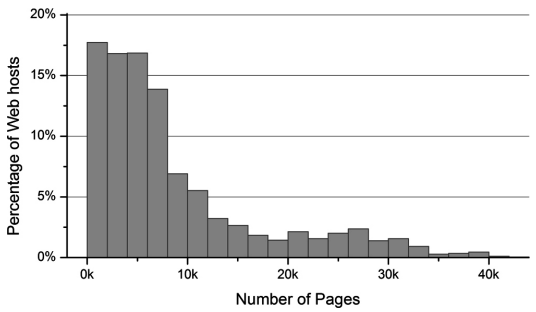


Fig. 14 Distribution of the page number.

Fig. 12 (a), most of the nodes still concentrate on four small areas. This severely uneven node distribution causes load balancing issue in the simulation. Figure 13 shows the relative error of the network coordinates.

The king dataset (1740 nodes) is divided into two sets: 1600 simulated Web hosts and 140 simulated crawlers. We randomly select 1600 hosts from the 1738 hosts used in 5.2 and map their sizes (number of pages, 10 KB each) to the 1600 simulated Web hosts (the distribution of page number is shown in Fig. 14). The total number of Web pages is 15,187,511. The simulation involves 10000 steps. In order to quickly push the system into a stable mode, we insert 70% crawlers in the first step and within the first 10 steps we submit all the Web hosts (pieces).

On one hand, the crawlers are added to the system according to the Poisson distribution ($\lambda = 0.1$). The living time of each crawler obeys the normal distribution $N(1000, 200)$ which means 95% crawlers' living time is be-

[†]http:// projects-deim.urv.cat/trac/planetsim/

[††]http://pdos.csail.mit.edu/p2psim/kingdata/

tween $1000 - 1.96 \cdot 200 = 608$ steps and $1000 + 1.96 \cdot 200 = 608$ steps. After the living time, the crawler either LEAVEs or FAILs with a random probability. Because we don't have too many unique crawlers (only 140) in the dataset, the dead crawlers are recycled: each dead crawler is re-inserted to the event chain and will re-JOIN the DHT later again. The re-JOIN time gap obeys the normal distribution $N(600, 200)$.

On the other hand, the Web hosts are submitted to the system according to the Poisson distribution (= 1) which is more frequent than that of the crawlers. We also simulate the situation when a Web host can be partitioned into pieces. Here, we use the partition results of the 1600 hosts (randomly selected from the 1738 hosts) to form the Web hosts' pieces. The number of pieces a Web host can be split are decided according to the overlap rate (less than 50$). Typically each Web host is split to 2 3 pieces. The maximum number of pieces of one Web host is no more than 4. All Web hosts and pieces are periodically re-crawled (re-submitted), the time gap of each re-crawl is fixed to 500 steps. During the crawl, we assign at most one TCP connection to each Web host (piece). The maximum number of concurrent TCP connections on each crawler is fixed to 5. If the number of Web hosts (or pieces) assigned to the crawler exceeds this number, they are queued. If a Web host (or piece) is queued on a crawler so long that the time exceeds the re-crawl time gap (500 steps), then the Web host (or piece) will be canceled by the crawler. We choose to set the number of concurrent TCP connections to a relatively small value so that the issue of load balancing can stand out.

All the crawler-host download rate are equally sized (1 Mb/s), while the crawler-host RTT are extracted from the king dataset. Then the execution time of each Web host is calculated by adding all the RTTs, DLTs and PWTs (100 ms) involved. The execution time (ms) of each Web host is mapped to steps by dividing the execution time by $2 \cdot 10^4$. For example, assuming that for one Web page the RTT = 100 ms, DLT = 100 ms, PWT = 100 ms, the total download time (execution time) of the Web host (or piece) with 10000 pages is $(100+100+100) \cdot 10000 = 3 \cdot 10^6$. Then the execution steps is $\frac{3 \cdot 10^6}{2 \cdot 10^4} = 150$. According to this mapping method, the total simulation time (10000 steps) equals to 55.56 hours. The crawler's memory size is fixed to 1 GB. Thus the crawler's capacity $min(BW_{download}, S_{memory})$ equals to $min(1\,Mb/s, 1\,GB) = 1$. And the crawler's RTL equals to $RTL_i = \frac{N_{pieces}}{1} = N_{pieces}$.

5.3.2 Simulation Results

In the simulation, we compare the following schemes:

DHT Conventional DHT: SHA1 hash-based Chord (the method of Apoidea) and CAN.

NC-CAN Proposed in Sect. 3.

LBI LBI-optimized NC-CAN proposed in Sect. 4. We set TTL = 3 (the best choice according to experience), OT = 11.

LBI_split LBI with partitioned Web hosts: we split larger Web hosts into pieces and submit the pieces in turn.

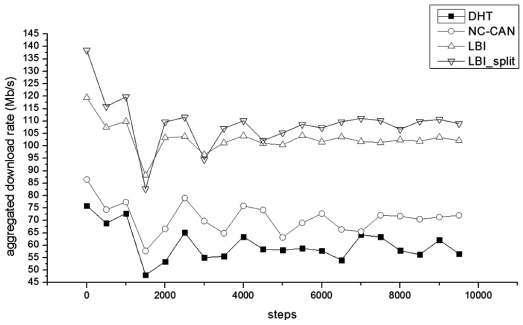


Fig. 15 System throughput comparison.

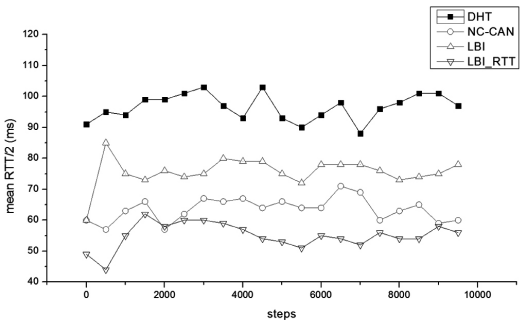


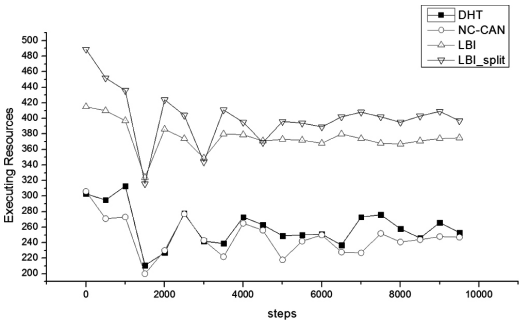
Fig. 16 RTT/2 comparison.

We found that with SHA1 hash, Chord and CAN performs similarly. As a result, only the performance of CAN is presented in the figures. The OT in *LBI* scheme is decided according to the number of the Web hosts (1600) and the number of the crawlers (140). On average the N_{pieces} is $\frac{1600}{140} = 11.4$; as the crawlers' capacities are equally sized, the OT equals to $\frac{11.4}{min(1,1)} = 11.4 \approx 11$

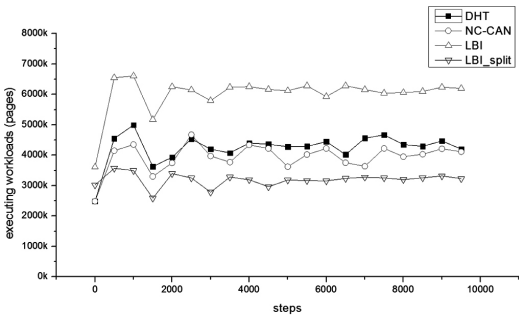
Figure 15 shows the aggregated download rate of the system (the combination of all crawlers' download rate) at different steps. We use the download rate as the metric to evaluate the system's throughput. On one hand, compared with conventional *DHT*, *NC-CAN* on average increases the system throughput by 27%. The result is mainly caused by the NC-estimated RTTs. As we analyzed in 5.1, reducing the crawler-host RTTs can reduce the download time of each Web page, thus the aggregated download rate can be increased. On the other hand, *LBI* and *LBI_split* achieve much higher download rate (more than 43% increase) and *LBI_split* outperforms *LBI* by 10%.

Figure 16 shows the mean of crawler-host RTT/2s. First, through both Fig. 15 and Fig. 16 we can see that the scheme with the highest RTTs which is *DHT* has the lowest system throughput. Second, although *NC-CAN* has lower RTTs than that of the *LBI*, the *LBI* has higher download rate. This time the determining factor is load balancing. The issue is further investigated in the following literature.

In addition, we add a trivial scheme called *LBI_RTT* to try to save LBI's RTT reduction. Under this method, when a crawler A is to forward a Web host a, crawler A

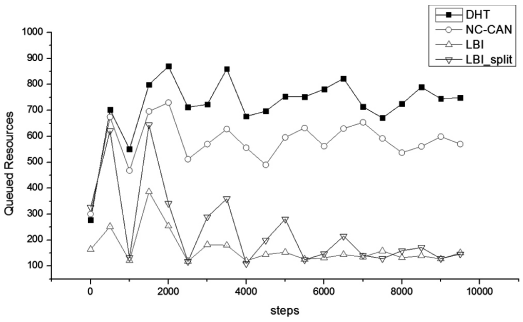


(a) executing tasks

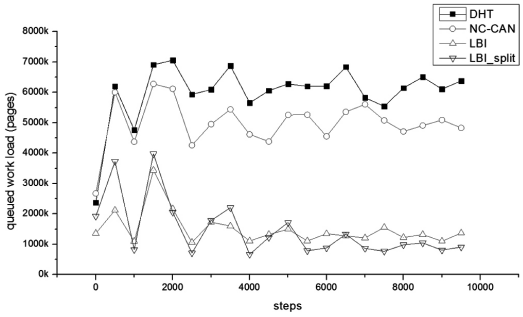


(b) executing workloads

Fig. 17 Executing tasks comparison.



(a) queued tasks



(b) queued work loads

Fig. 18 Queued tasks comparison.

not only compares the RPL of its neighboring crawlers but also compares the Euclidean distances between each neighboring crawler and a. Crawler A then derives a forwarding target by weighing the two factors. This method achieves even lower RTTs than *NC-CAN*. However, its throughput is very similar to that of the *LBI* (the two curves almost coincide so we don't draw it in Fig. 15).

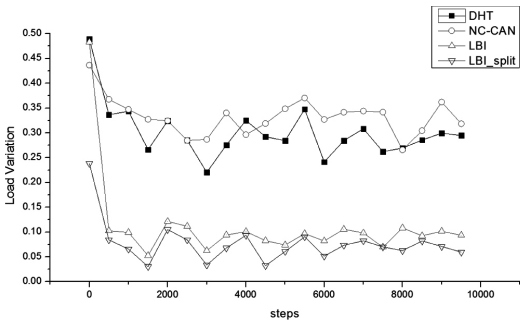
Figure 17 (a) shows the number of executing tasks within the system. Here we treat the Web hosts and the pieces equally as tasks because from the perspective of a crawler they don't have substantial difference. The curves is very similar to Fig. 15. Intuitively, we can conclude that the more tasks the system is executing the more download rate the system can achieve. However, this is only partially true as the curve of *NC-CAN* is even lower than the curve of the *DHT*. The reason why the download rate of *NC-CAN* still exceeds that of the *DHT* in Fig. 15 lies in the RTTs. With lower RTTs, the crawlers in the *NC-CAN*-system can achieve higher download rates. The download rates of *NC-CAN*-scheme are so high that they catch up with the *DHT*-scheme which has more executing tasks.

In Fig. 17 (b) we take the number of pages (we call the workloads) in each Web host into consideration. An important thing to note is that the curve of the *LBI_split* is very low. This is because the Web hosts are split into pieces. And the workload of each piece is relatively small. In the following analysis we can see that smaller workload leads to shorter execution time which is important in a dynamic environment where crawlers JOIN and LEAVE (FAIL) constantly.

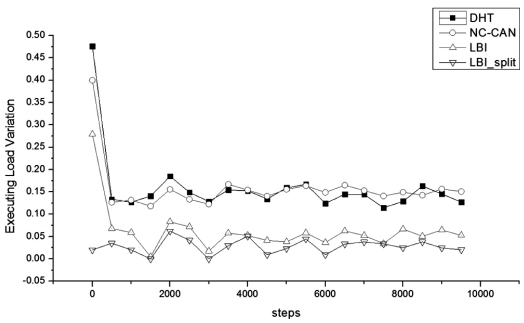
Figure 18 (a) shows the number of queued tasks. Figure 18 (b) takes the number of pages (we call the workloads) in each Web host into consideration. In Fig. 18 (a) the *LBI_split*'s queued tasks outnumber the *LBI*'s. This is because there are more tasks (pieces) within the *LBI_split*-system. However, when it comes to the workloads, we can see that the *LBI_split*-system doesn't queue more workloads than the *LBI*-system does. The figures also reveal why there are more tasks under execution in the *LBI**-system: more than half of the tasks in *DHT*-system and *NC-CAN*-system are queued! We think this is because a portion of crawlers in these systems bears too many tasks (and workloads) while the other crawlers are almost idle. The analysis of Fig. 19 confirms our assumption.

Figure 19 shows the coefficient variation of each crawler's tasks (we call load variation). In Fig. 19 (a), we take all the tasks assigned to the crawlers (including executing tasks and queued tasks) into consideration. As is already shown in Fig. 12, the Web hosts unevenly distribute across the logical space. This phenomenon causes load balancing issue. *NC-CAN*-scheme in Fig. 19 (a) performs the worst because it doesn't make adjustment to the issue. *DHT*-scheme performs a little better, because it uses a SHA1 hash method instead of the network coordinates. In contrast, all the *LBI** methods leads to much lower load variation. The *LBI_split* shows the lowest load variation because 1) it has smaller task granularity and 2) the number of tasks (pieces) within the system is bigger than that of the *LBI*. In Fig. 19 (b) we only take the executing tasks into consideration. The trend is very similar to Fig. 19 (a).

In our system, if a crawler LEAVEs or FAILs, the tasks assigned to it are migrated to other crawlers. The dead

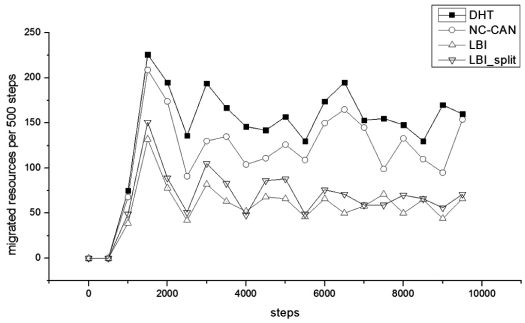


(a) all tasks

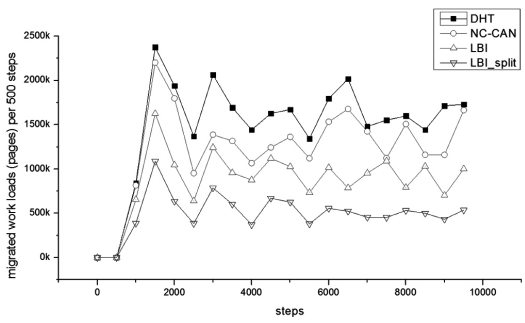


(b) executing tasks

Fig. 19 Load variation comparison.



(a) migrated tasks



(b) migrated work loads

Fig. 20 Migrated tasks comparison.

crawler's successor re-executes all the inherited tasks. An alternative design (we call the recovery strategy) is to periodically replicate the crawler's breakpoint information (including the current depth, current URL todo and URL seen data structures) and recover the tasks accordingly when the crawler leaves the system. We consider the re-execution strategy a better choice than the recovery strategy since periodically replicating breakpoints brings more additional bandwidth cost to the system. Figure 20 (a) shows the number of migrated tasks and Fig. 20 (b) shows the number of migrated workloads recorded under the re-execution strategy. The number of *NC-CAN*-system is smaller than that of *DHT*-system, because compared with the *DHT*-system tasks within the *NC-CAN*-system have shorter execution time (download time). But because there are too many tasks queued in the *NC-CAN*-system, many queued tasks are migrated even before their execution. In contrast, thanks to the load balancing, more tasks are able to be completed before their crawlers leave the system. From Fig. 20 (b) we can see that *LBI_split* has the smallest number of migrated workloads which means that the time cost on re-executing the migrated tasks is also small.

More re-crawl times leads to higher update-rate which is very important to a Web search service. As the Web hosts (or pieces) are periodically submitted to the crawlers (every 500 steps), ideally, each Web host should be re-crawled 20 times (10000 steps/500 steps). However, in our simulation, we find that not all the Web hosts are re-crawled that many times. Some tasks are cancelled because they have been waiting in the crawler's queue for over 500 steps. We draw a CDF (Fig. 21) to show the distribution of each Web host's

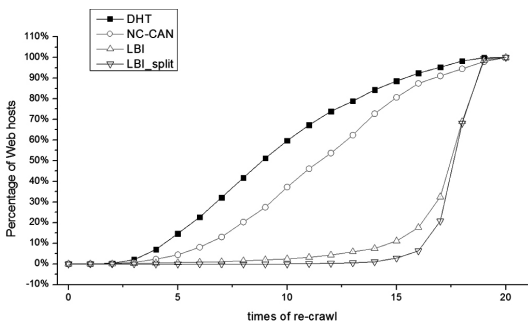
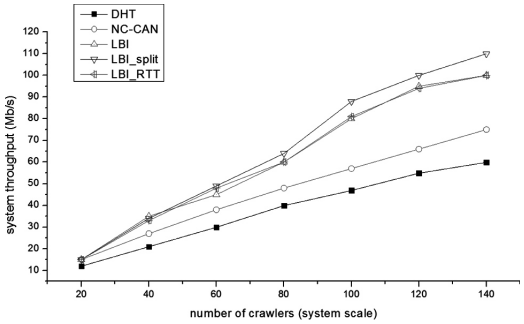


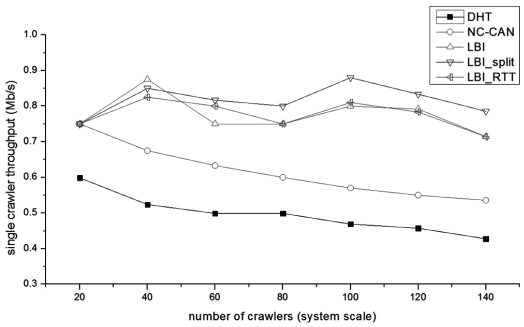
Fig. 21 Execution time comparison (CDF).

re-crawl times. From the figure we can see that all the Web hosts under the four system schemes are at least re-crawled 2 times. More than 50% Web hosts within *NC-CAN*-system are re-crawled more than 10 times. More than 90% Web hosts within the *LBI**-systems are re-crawled more than 15 times. The result means that each Web host is re-crawled (updated) every 3.7 hours under *LBI**-systems.

The final experiment we take is a scalability test. Here we separately run 7 tests for each scheme we mentioned before. In each test, the crawlers' maximum number is limited to 20, 40, 60, 80, 100, 120 and 140. Figure 22 (a) shows the statistics on the system throughput. Figure 22 (b) shows the average throughput of each single crawler. Under *LBI**-scheme, the single-crawler throughput always maintains between 0.75 Mb/s and 0.9 Mb/s. On the other hand, the crawler's throughput under *NC-CAN*-scheme and *DHT*-scheme tends to decrease. Both *NC-CAN*-scheme and *DHT*-scheme decrease 0.15 Mb/s of throughput when the num-



(a) system throughput



(b) single crawler throughput

Fig. 22 Scalability comparison.

ber of crawlers changes from 20 to 140. Nevertheless, *NC-CAN*-scheme still outperforms DHT-scheme by about 30%. The results shows the high scalability of the *LBI**-schemes.

5.3.3 Conclusions

Here we conclude the result of our simulations. The system's performance is mainly contributed by the two factors: the crawler-host RTTs and load variation. The conventional DHT-based system model (such as Apoidea) only achieves conservative performance since it barely pushes the load balancing issue to the hash function and ignores the impact of the RTTs. In our simulation, the optimization on the RTTs described in 3.2 (*NC-CAN*) increases the system's throughput by 27%. However, through a further analysis we find that the system's performance can be further optimized by balancing the load among the crawlers. By applying the load balancing method proposed in 4.2 (*LBI*), the system's throughput can increase more than 43%. We also test our Web host partition method (*LBI_split*). The method also brings improvements to the *LBI*. Finally, we take a scalability test which shows the high scalability of the *LBI**-methods.

6. Related Works

The existing distributed Web crawling systems can be classified into two types: the cluster-based systems and the Internet-based systems. The cluster-based systems such as Mercator [25], the early form of Google [26] have their machines located in the same location both in the network and the physical world. There's no way to reduce network distance between the crawlers and hosts since all crawlers subject to the same network distance to any Web site.

The Internet-based systems have their machines located all over the Internet, so that they are able to make optimizations in reducing the crawler-host distance and achieve a higher retrieving speed. The research on the Internet-based crawling generally started from the beginning of the 21st century, as the cluster-based systems are facing bottlenecks in network bandwidth, update rate and scalability. Cho, J. [11] first proposed a series of basic concepts on paralle crawling and distributed crawling, including classification methods, evaluation metrics and so on. After 2003, several systems were proposed: UbiCrawler [6] is the first crawling system announced to be deployed on WAN. The consistent hashing [7] method it adopts guarantees the load balancing among crawlers. However, it prevents the system from optimizations on network distance. In IPMicra [8], crawler is selected to crawl a certain Web site if they were located in the same AS or ISP network according to the information provided by Regional Internet Registries (RIRs); SE4SEE [27] reduces the network distance by assigning crawler Web sites that were located within the crawler's country; Apoidea [8] implements a Chord-based DWC system. However, it didn't make optimizations in reducing the crawler-host distance.

Many Internet-based systems are also built by commercial companies and open source community, but few detailed information was exposed about these systems. So far as we know, YaCy [3] is a distributed search engine providing user-customized search services; Majestic [5] implemented a crawling system using the architecture of SETI@Home [2] and sets up a top list of contributors; FAROO [4] implemented a fully distributed P2P search engine and feeds back profit to the resource contributors.

7. Conclusions and Future Works

DHT-based distributed Web crawling (DWC) is proposed to implement the Web search service over ordinary computing resources across the Internet. Content Addressable Network (CAN) over Network Coordinate System (NC) provides a way to map network distances to logical distances on the DHT. Based on this concept, we propose a new system model for DHT-based distributed Web crawling. We also propose a new load balancing method for CAN in which the load balancing operation is performed when each resource (Web host) is inserted (submitted) into the DHT. Through our small scale tests, we not only prove the efficiency of our system model but also find that the crawler-host RTT plays an important role in determining the system's total throughput. Through a larger scale simulation, we further evaluate the system's performance.

We recognize that in our current system model, the NC and manager may become the system's bottleneck as the number of Web hosts and crawlers continually increase. In the future, we want to find more scalable solutions and evaluate the system's scalability.

The other problem is that if a node (crawler) doesn't want to do the jobs it could always pretend to be overloaded and send the jobs to the other nodes, which increases the possibility of selfishness. We consider the selfishness can be avoided if the selfish behavior brings no benefit. A reasonable way is to build a mechanism to encourage the contribution of computing resources, e.g. feedback profit (from the search service) to the contributor according to the number of Web pages his/her machine has downloaded. Under this mechanism, the selfish nodes who don't do the jobs would gain no profit at all. Therefore, the selfish nodes have no reason to be "selfish".

Currently we only implement the functionality of crawling the Web 1.0 sites, for example news sites and university sites. We recognize our system model can be applied to the BBS site. The only difference is the design of the crawler since crawling the BBS sites needs some additional functionality such as login. To the Blog sites we currently don't have sufficient solutions since these sites may consist of millions of independent hostnames and lack global indexes. In the future, we want to scale our system to support the BBS sites and Blog sites.

Acknowledgments

I would like to give my special thanks to Master students Li Sun, Chuan-Liang Yu and Yi-Fan Wei who helped me a lot during the writing of this paper.

This paper was partially supported by the National Natural Science Foundation of China under Grant No.60703014; the National Grand Fundamental Research 973 Program of China under Grant No.G2005CB321806; the Specialized Research Fund for the Doctoral Program of Higher Education, SRFDP No. 20070213044; the National High-Tech Research and Development Plan of China under Grant Nos. 2007AA01Z442; the National High-Tech Research and Development Plan of China under Grant Nos. 2009AA01Z437.

References

[1] I. Foster, "Internet computing and the emerging grid," Nature, 2000.
[2] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela, "Seti@home–massively distributed computing for seti," Comput. Sci. Eng., vol.3, pp.78–83, 2001.
[3] "Yacy distributed Web search." http://yacy.net/
[4] "Faroo real time searchh." http://www.faroo.com/
[5] "Majesti-12: Distributed Web search." http://www.majestic12.co.uk/
[6] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed Web crawler," Eighth Australian World Wide Web Conference (AUSWEB'02), 2002.
[7] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide Web," ACM Symposium on Theory of Computing, pp.654–663, ACM, New York, NY, USA, 1997.
[8] A. Singh, M. Srivatsa, L. Liu, and T. Miller, "Apoidea: A decentralized peer-to-peer architecture for crawling the world wide Web," SIGIR'03 Workshop on Distributed Information Retrieval, pp.126–142, ACM, 2003.
[9] O. Papapetrou and G. Samaras, "Ipmicra: Toward a distributed and adaptable location-aware Web crawler," ADBIS'04, Budapest, Hungary, 2004.
[10] S. Ratnasamy, P. Francis, R.K.M. Handley, and S. Shenker, "A scalable content addressable network," ACM SIGCOMM'01, pp.161–172, ACM, San Diego, California, USA, 2001.
[11] J. Cho and H. Garcia-Molina, "Parallel crawlers," 11th International Conference on World Wide Web, pp.124–135, ACM, New York, NY, USA, 2002.
[12] H.Z.T.S.E. Ng, "Towards global network positioning," ACM SIGCOMM Internet Measurement Workshop, San Francisco, CA, USA, 2001.
[13] H.Z.T.S.E. Ng, "A network positioning system for the Internet," USENIX Annual Technical Conference, Berkeley, CA, USA, 2004.
[14] M.C.A.R.M. Costa, "Pic: Practical Internet coordinates for distance estimation," International Conference on Distributed Systems, Tokyo, Japan, 2004.
[15] J.C.S.W.M. Pias, "Lighthouses for scalable distributed location," 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), 2003.
[16] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," ACM SIGCOMM'04, pp.15–26, 2004.
[17] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, and I. Stoica, "The impact of dht routing geometry on resilience and proximity," ACM SIGCOMM'03, pp.381–394, 2003.
[18] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," INFOCOM'02, 2002.
[19] K. Doi, S. Tagashira, and S. Fujita, "Proximity-aware content addressable network based on Vivaldi network coordinate system," 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2002.
[20] O.D. Sahin, A. Gupta, D. Agrawal, and A.E. Abbadi, "A peer-to-peer framework for caching range queries," ICDE'04, 2004.
[21] P. Ganesan, B. Yang, and H. Garcia-Molina, "One torus to rule them all: Multi-dimensional queries in p2p systems," 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS2004, pp.19–24, New York, NY, USA, 2004.
[22] A. Gupta, O.D. Sahin, D. Agrawal, and A.E. Abbadi, "Meghdoot: Content-based publish/subscribe over p2p networks," 5th ACM/IFIP/USENIX international conference on Middleware, pp.254–273, 2004.
[23] J. Xiong, Q. Qi, P. Hong, and J. Li, "A few optimized load balancing methods of content addressable network," J. Electronics & Information Technology, 2006.
[24] D. Takemoto, S. Tagashira, and S. Fujita, "Distributed algorithms for balanced zone partitioning in content-addressable networks," 10th International Conference on the Parallel and Distributed Systems, Washington, DC, USA, 2004.
[25] A. Heydon and M. Najork, "Mercator: A scalable, extensible Web crawler," World Wide Web, vol.2, pp.219–229, 1999.
[26] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," Computer Networks and ISDN Systems, vol.30, pp.107–117, 1998.
[27] B.B. Cambazoglu, E. Karaca, T. Kucukyilmaz, A. Turk, and C. Aykanat, "Architecture of a grid-enabled Web search engine," Inf. Process. Manage., vol.43, pp.609–623, 2007.

Xiao Xu received his B.S. and M.S. degrees in computer science from Harbin Institute of Technology (HIT) in 2005 and 2007. He is now working towards his Ph.D. degree in computer science at Harbin Institute of Technology. His research interests include computer network, distributed computing.

Weizhe Zhang received his B.S., M.S. and Ph.D. degrees in computer science from Harbin Institute of Technology (HIT) in 1999, 2001 and 2006 respectively. He has been working in HIT from 2002 and been an Associate Professor at computer science of HIT since 2007. His research interests include computer network, theory of computation and parallel computing.

Hongli Zhang received her Ph.D. degree in computer science in 1999. She is now a Professor of school of computer science and technology of Harbin Institute of Technology (HIT). She is a member of China Computer Federation. Her research interests include network security and grid computing.

Binxing Fang received his Ph.D. in computer science in 1989. He is a Professor of school of computer science and technology of Harbin Institute of Technology (HIT). He has been a Academician of Chinese Academy of Engineering since 2005. His research interests include network security and grid computing.